Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT202-008-S2024/it202-api-project-milestone-2-2024/grade/jhr4

IT202-008-S2024 - [IT202] API Project Milestone 2 2024

Submissions:

Submission Selection

1 Submission [active] 4/12/2024 9:45:03 PM

Instructions

COLLAPSE

Implement the Milestone 2 features from the project's proposal document: https://docs.google.com/document/d/1XE96a8DQ52Vp49XACBDTNCq0xYDt3kF29cO88E
Make sure you add your ucid/date as code comments where code changes are done
All code changes should reach the Milestone2 branch
Create a pull request from Milestone2 to dev and keep it open until you get the output PDF from this assignment.
Gather the evidence of feature completion based on the below tasks.
Once finished, get the output PDF and copy/move it to your repository folder on your local machine.
Run the necessary git add, commit, and push steps to move it to GitHub
Complete the pull request that was opened earlier
Create and merge a pull request from dev to prod
Upload the same output PDF to Canvas

Branch name: Milestone2

Tasks: 29 Points: 10.00

Define Appropriate Tables for Data (1 pt.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshots of Table SQL

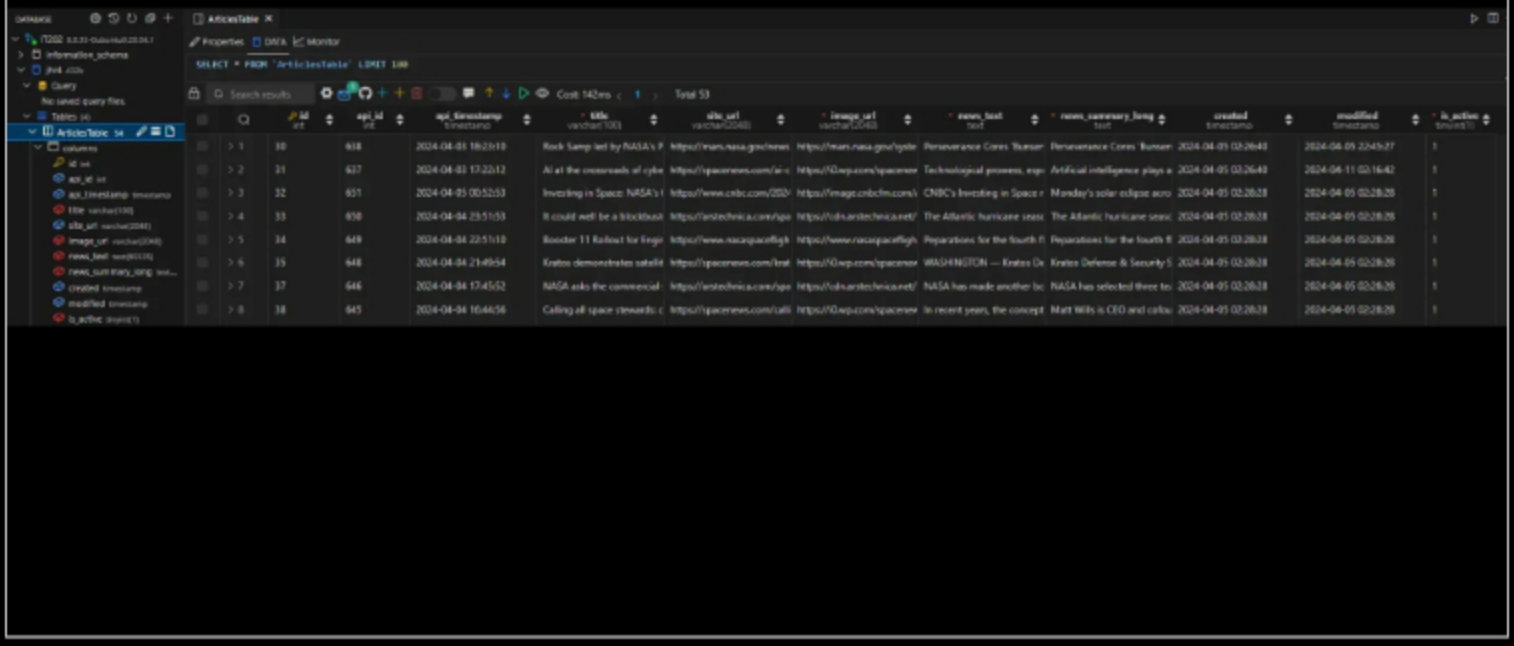
Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Table(s) should have the 3 core columns we'll always be using (id, created, modified) plus additional columns for the incoming API data
#2	1	Columns should be logical and thought out (not valid to have a single field of JSON data or similar
#3	1	Clearly caption screenshots

Task Screenshots:

Gallery Style: Large View

Small Medium Large



This is my table. It has the 3 core columns: id, created, and modified. It also has api_id, api_timestamp, title, site_url, image_url, news_text, and news_summary_long for the data from the api (or manual create, however for manual some values will be null like api_id). The is_active column was added for the soft delete of articles.

Checklist Items (3)

#1 Table(s) should have the 3 core columns we'll always be using (id, created, modified) plus additional columns for the incoming API data

#2 Columns should be logical and thought out (not valid to have a single field of JSON data or similar

#3 Clearly caption screenshots

COLLAPSE

Task #2 - Points: 1

Text: Explain the design

Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Note the different fields and their purpose
#2	1	Note if you needed to normalize the data into separate tables or if one table fit your needs

Response:

In the table I have the following fields/columns: id, api_id, api_timestamp, title, site_url, image_url, news_text, news_summary_long, is_active, created, and modified.

Id, created, and modified are there to show the time the articles were uploaded or edited and the id serves as a way to identify all the rows by a number.

The api_id is similar to the "id," but is more so to make sure no duplicates are passed through by the api as it has the 'unique' requirement. For non api articles "null" is passed in the api_id. The title was also made to unique just to make sure user created articles are also not duplicates. The api_timestamp is also like created, but is more so for the original articles created time which comes from the api. The title, site_url, image_url, news_summary_long, and news_text are the main components of the actual article which is why they are taken from the api. Readers should know where the article is from, what the article is about and the actual article plus a brief summary. The is_active column serves as a soft delete for the articles. For all these values one table was sufficient and no additional table was required.

COLLAPSE

Task #3 - Points: 1

Text: Add the pull request link for the branch related to this feature

Details:

Note: the link should end with /pull/#. Same pull request shouldn't be used for each feature.

URL #1

https://github.com/Jhr-4/jhr4-IT202-008/pull/54

COLLAPSE

Data Creation Page (2 pts.)

COLLAPSE

Task #1 - Points: 1

Text: Screenshots of the creation page

Checklist *The checkboxes are for your own tracking

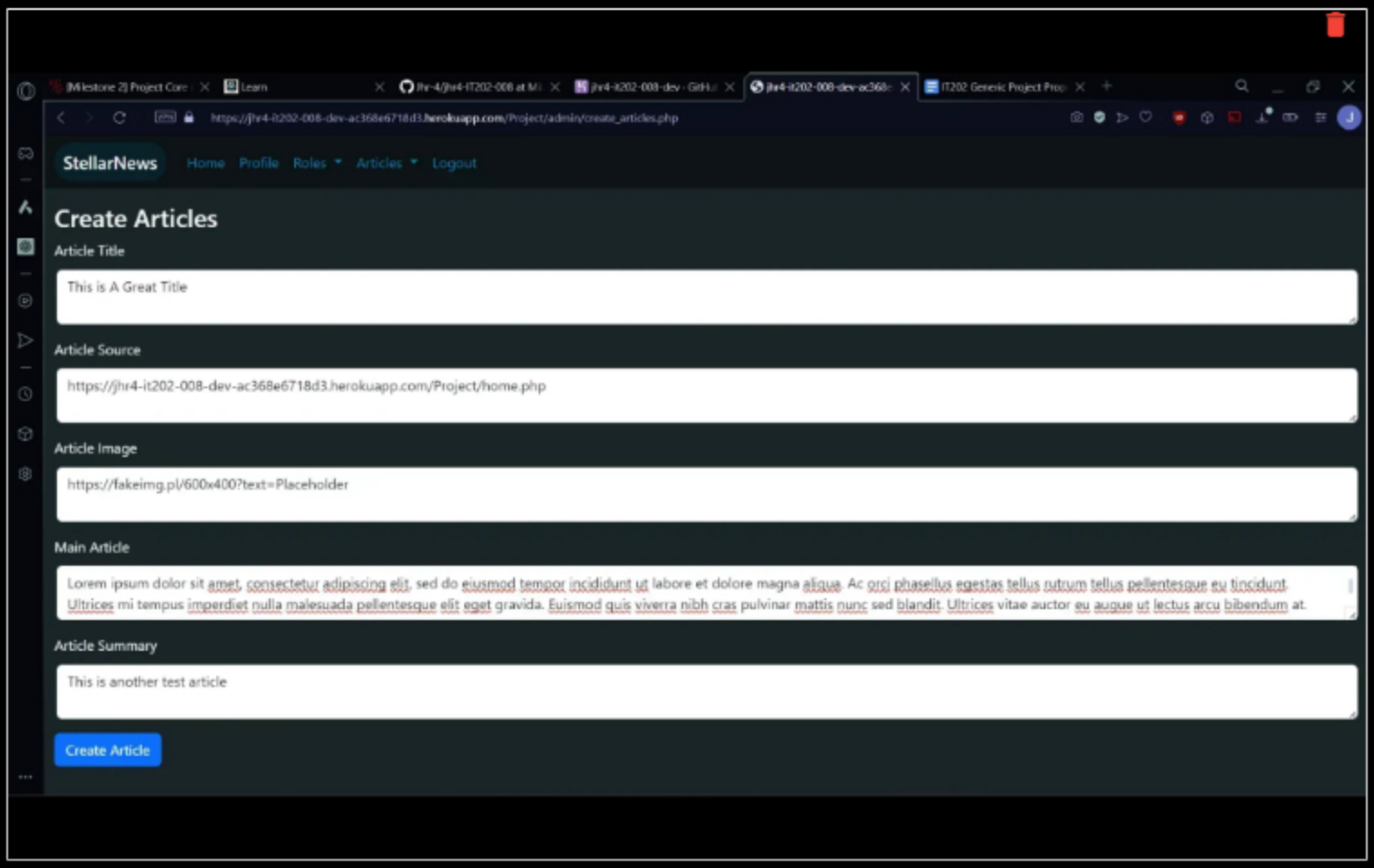
#	Points	Details

#1	1	Show potentially valid data filled in for the custom creation page
#2	1	Show how the API data is fetched for API data (must be server-side)
#3	1	Show examples of validation messages
#4	1	Show an example of successful creation message
#5	1	Design/Style should be considered (i.e., bootstrap, custom css, etc)
#6	1	Make sure the heroku dev url is visible in the address bar
#7	1	Clearly caption screenshots

Task Screenshots:

Gallery Style: Large View

Small Medium Large



Shows potentially valid data for the manual creation page. Heroku Dev URL on top. Design/Style is simple, but works. It's user friendly, made with bootstrap input fields. Heroku Dev Link on Top.

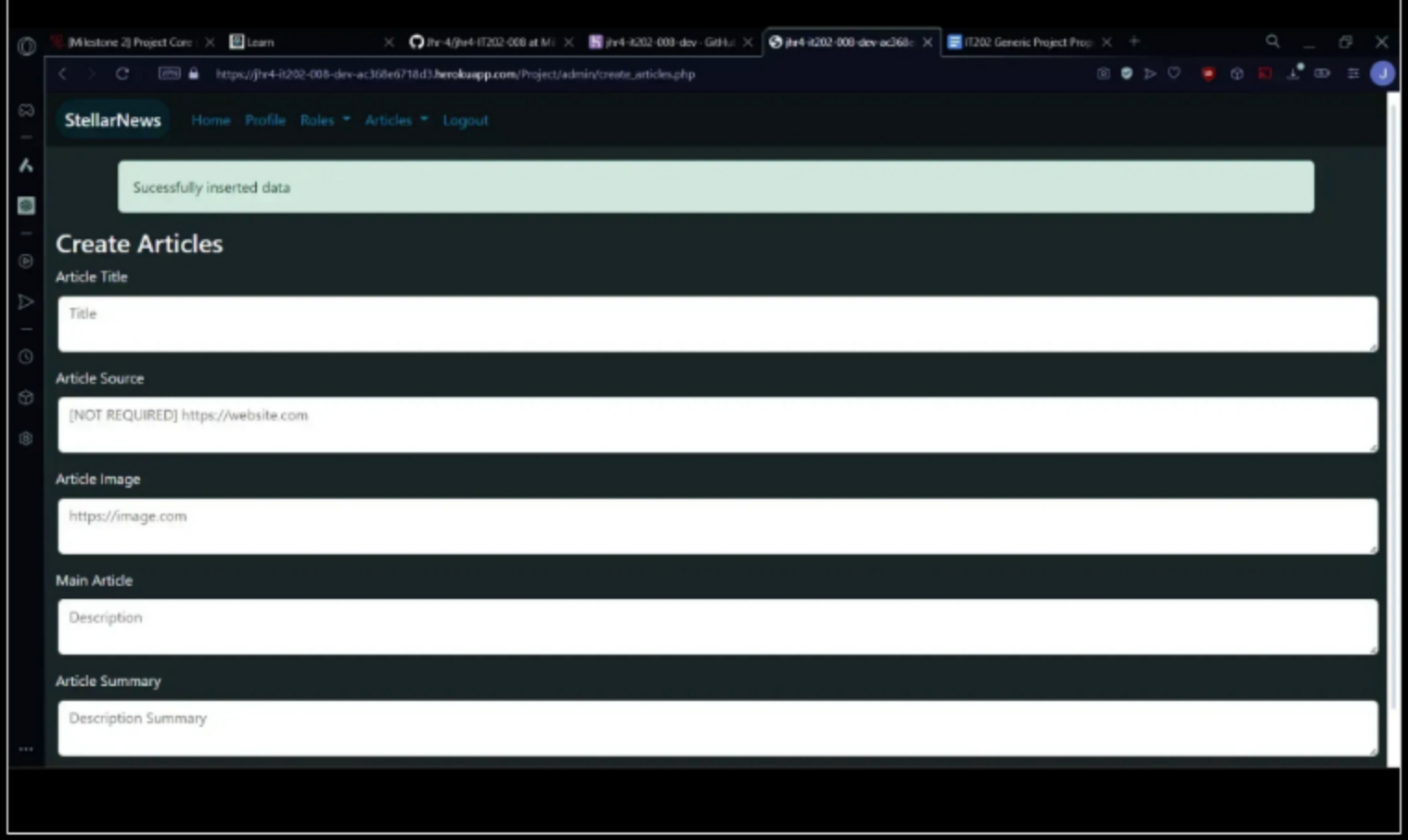
Checklist Items (4)

#1 Show potentially valid data filled in for the custom creation page

#5 Design/Style should be considered (i.e., bootstrap, custom css, etc)

#6 Make sure the heroku dev url is visible in the address bar

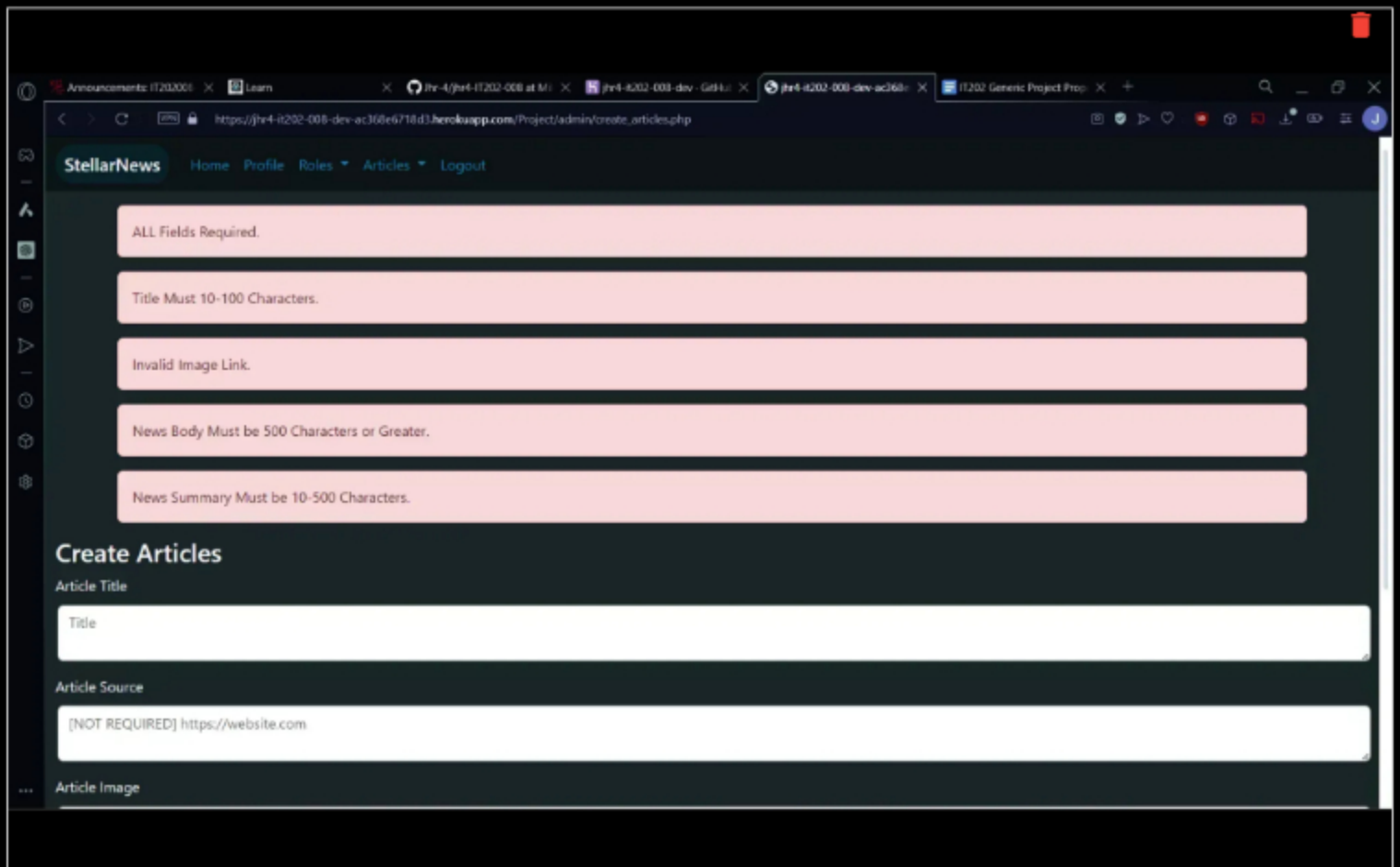
#7 Clearly caption screenshots



Upon pressing create, the successfully inserted data is shown indicating article has been created. Heroku URL on top.

Checklist Items (1)

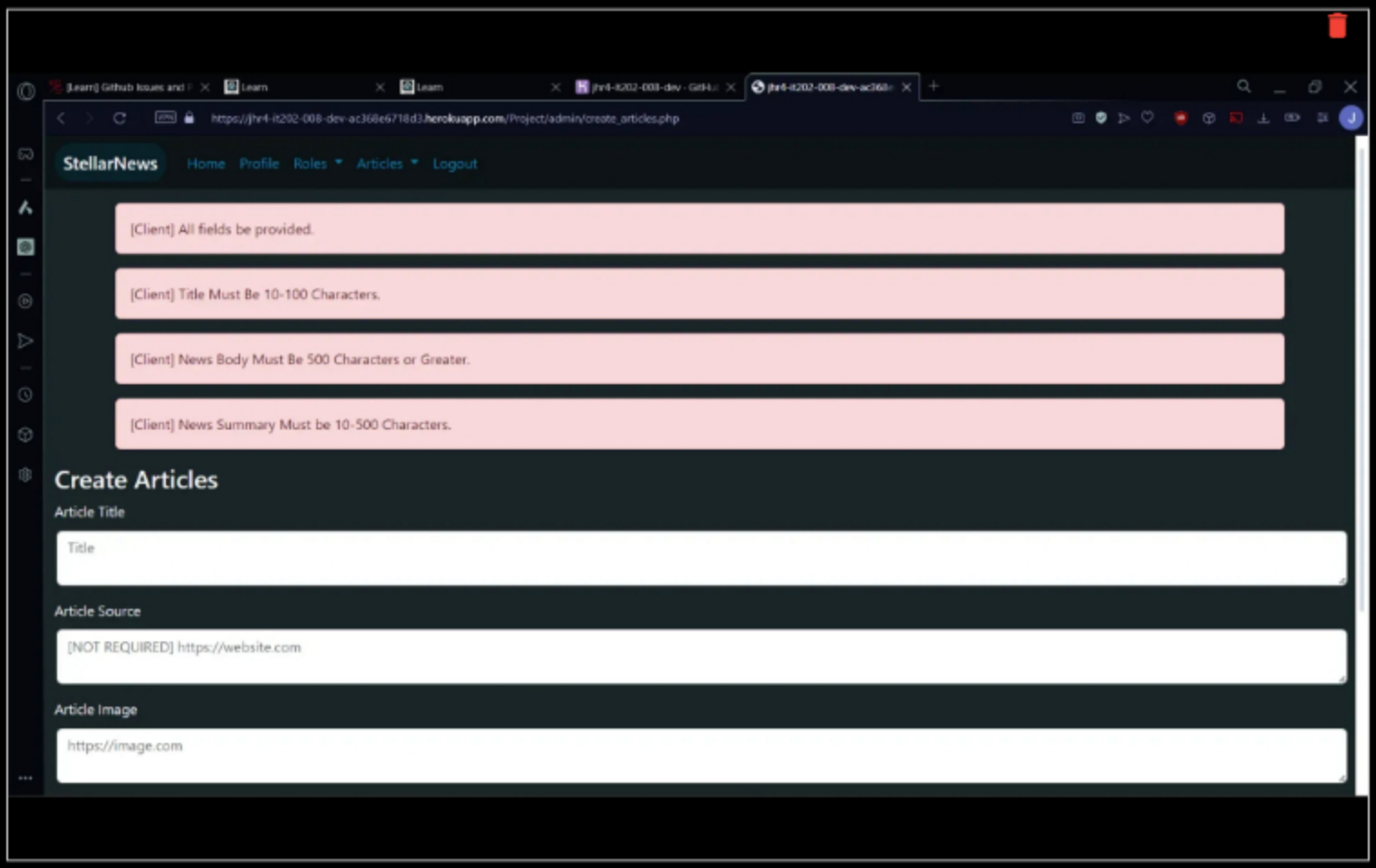
#4 Show an example of successful creation message



Server-Side PHP Validation. (this was seen because there was an error with JS validation which has been fixed as seen in the next screenshot, but it shows how the server side validation is working regardless of the error on client side.)

Checklist Items (1)

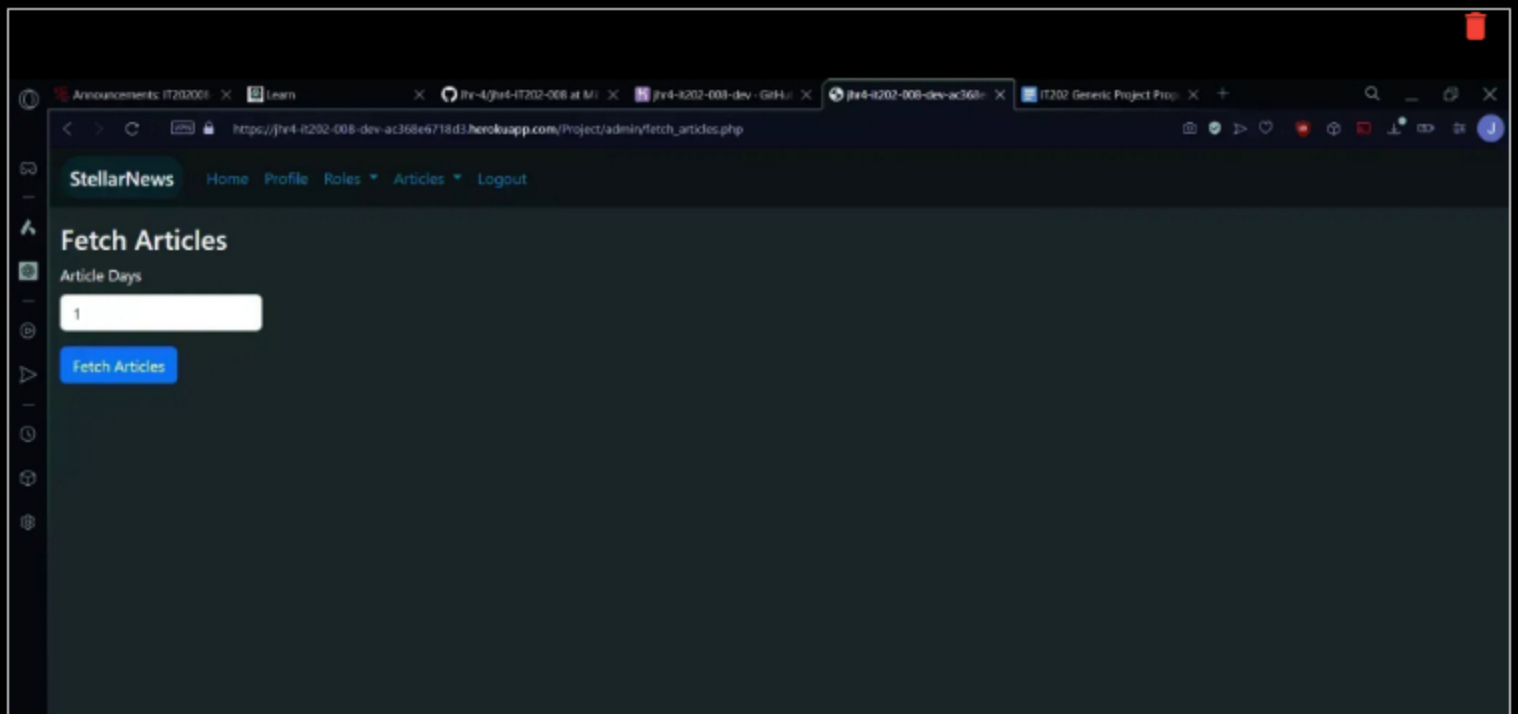
#3 Show examples of validation messages



Client Side JS Validation Messages.

Checklist Items (1)

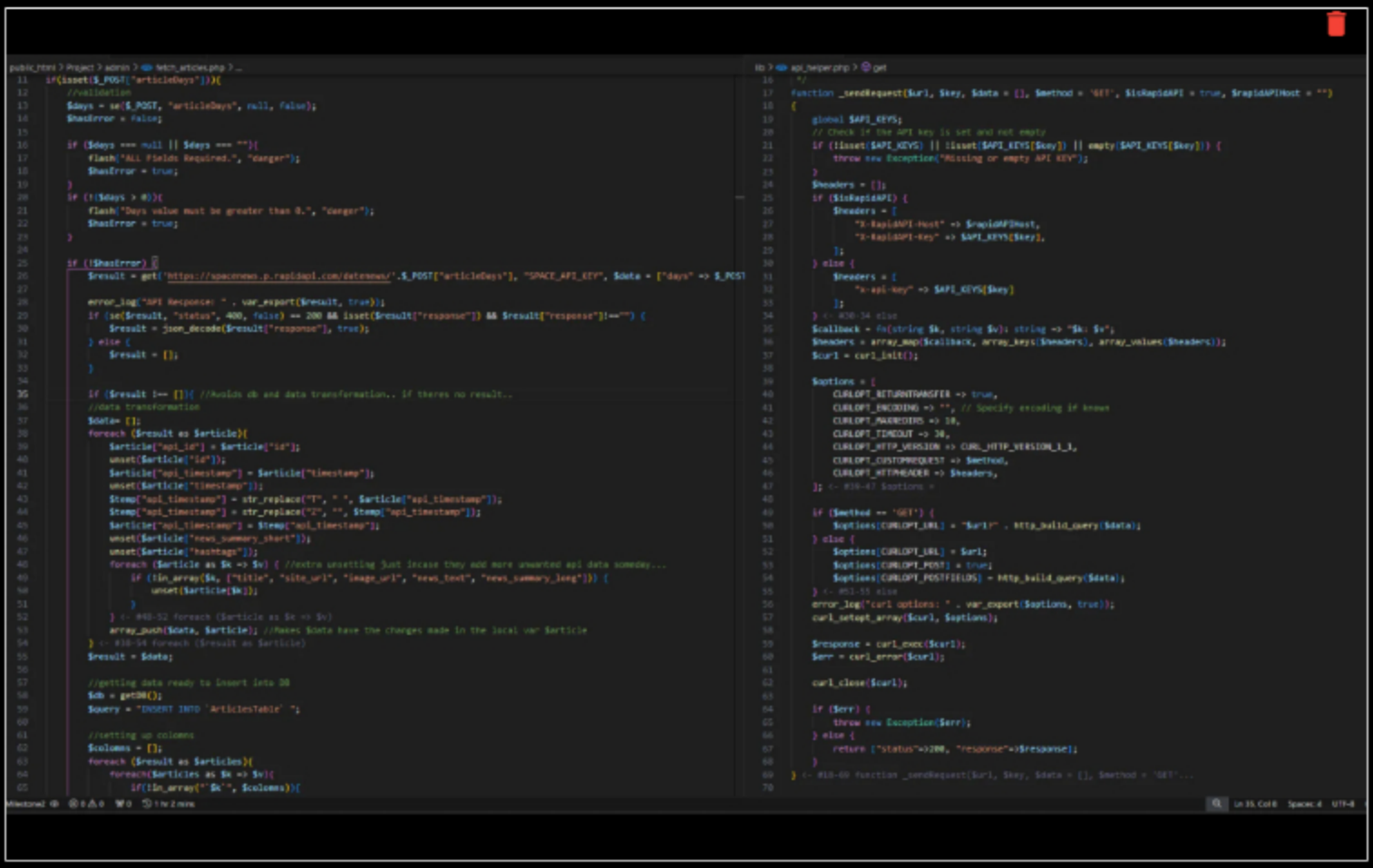
#3 Show examples of validation messages



API data fetched for API data via this form that asks for days input. Takes a value of minimum 1 or higher (has HTML, PHP, and JS validation).

Checklist Items (1)

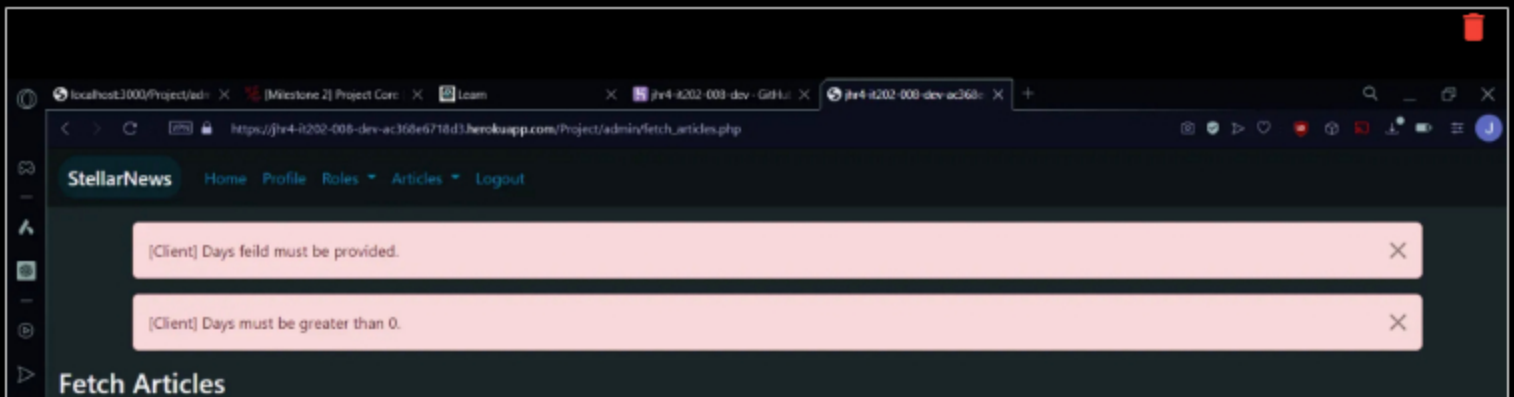
#2 Show how the API data is fetched for API data (must be server-side)

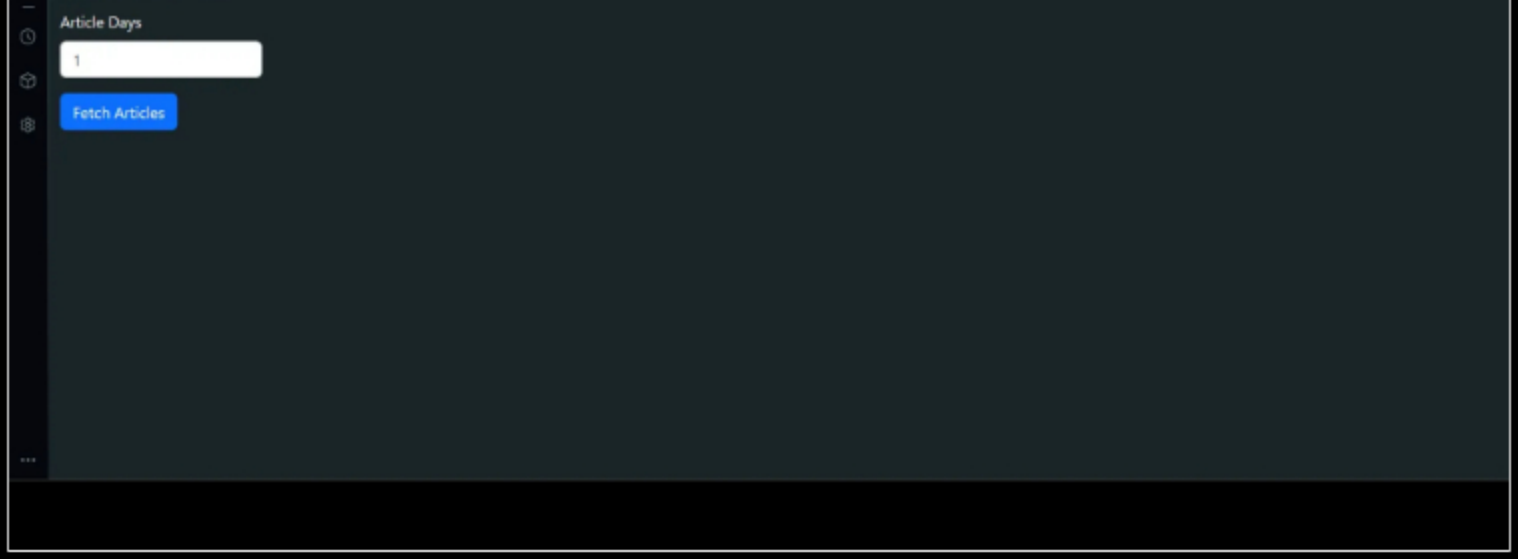


Shows how API data is fetched (Server-Side Code). The actual fetching logic is shown on the right side in which a curl is used to connect with the API and get the response which is checked if it was successful or not via the status code.

Checklist Items (1)

#2 Show how the API data is fetched for API data (must be server-side)

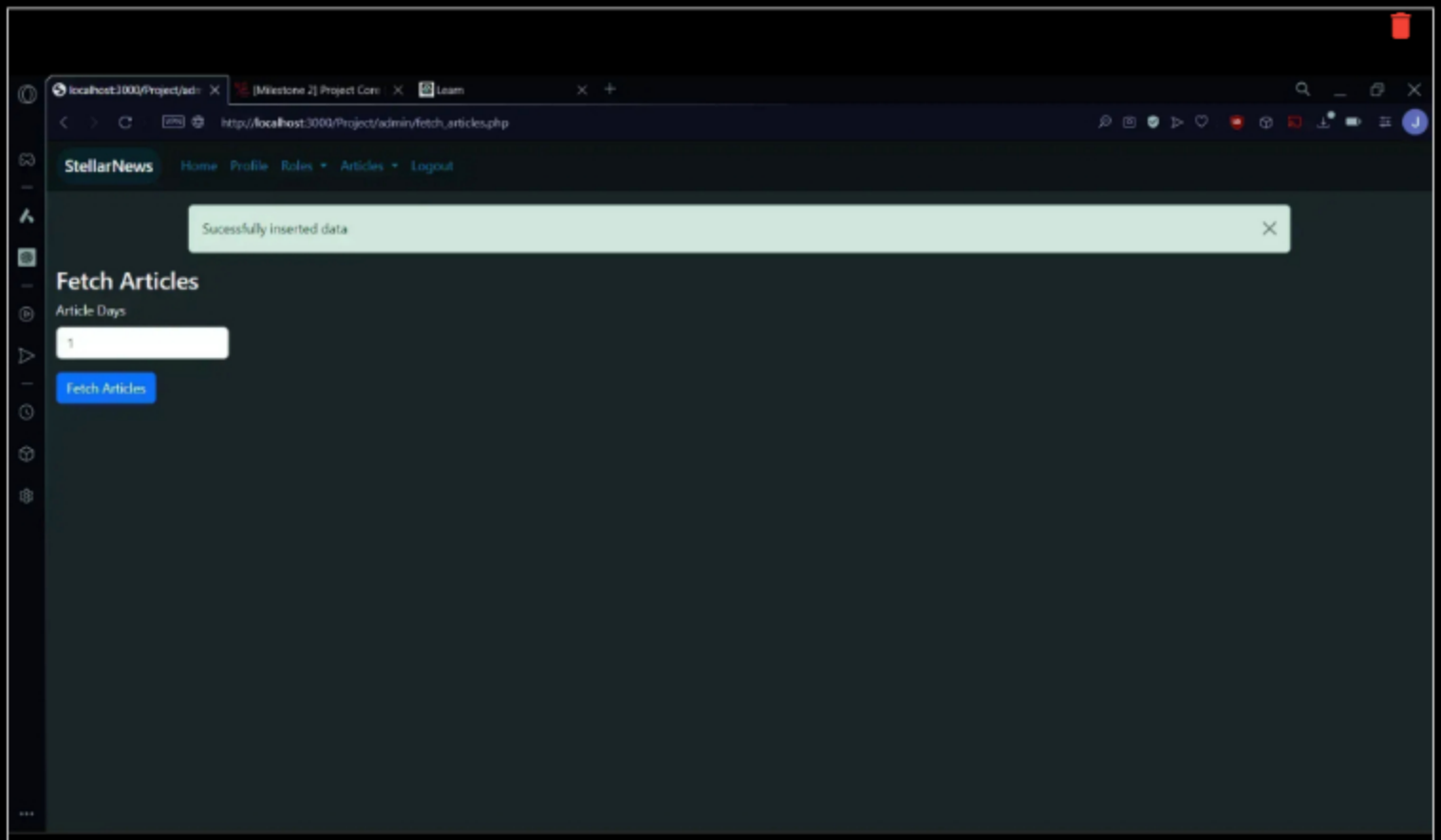




JS Validation Message for Fetch articles.

Checklist Items (1)

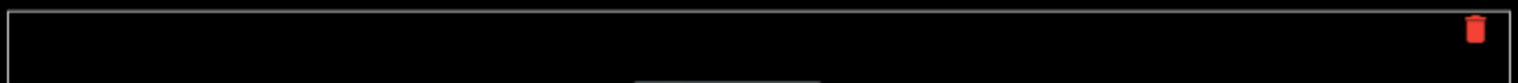
#3 Show examples of validation messages

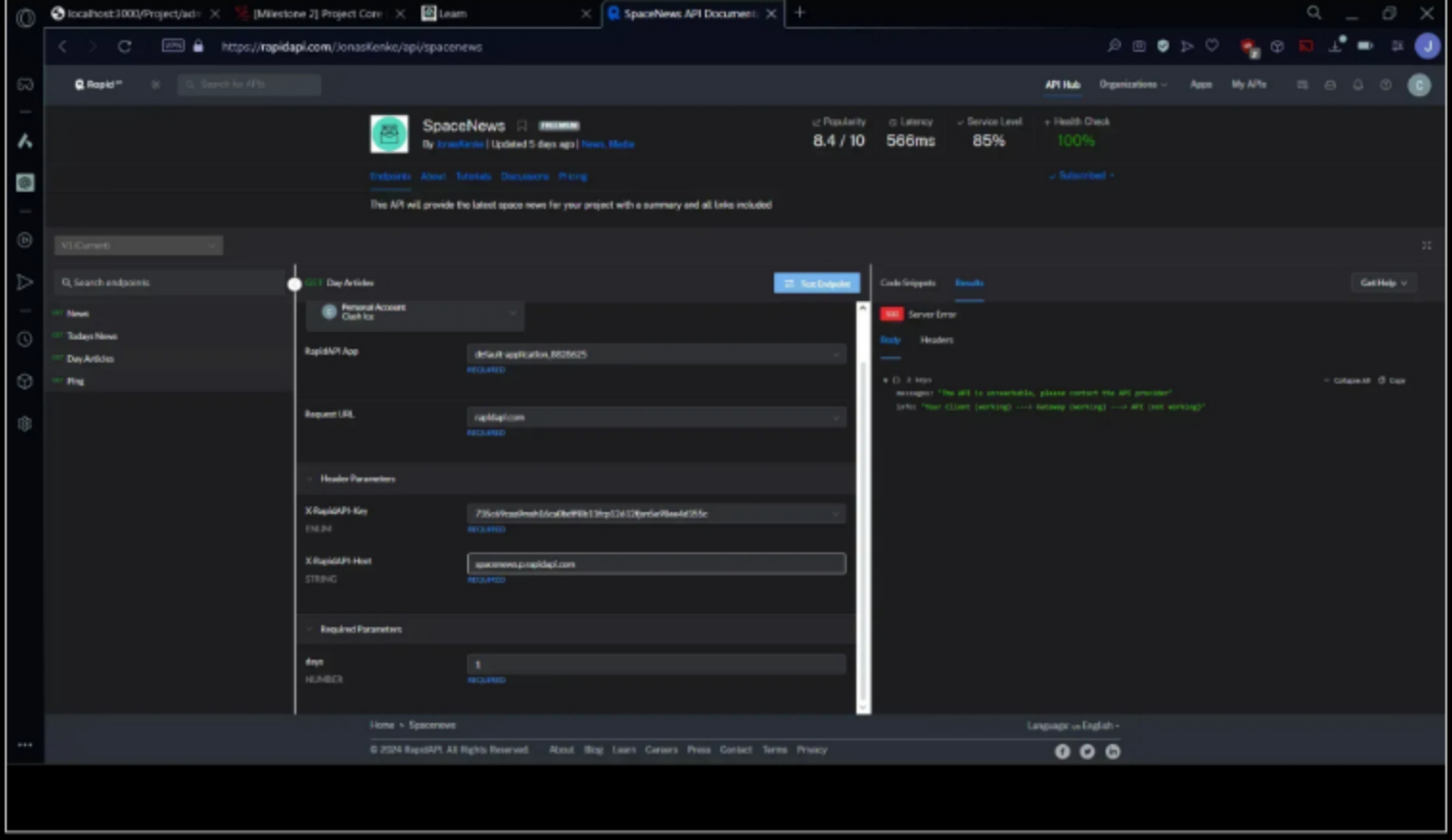


LocalHost: Successful Fetch Message utilizing cached data from earlier (from testing api with the api). The actual fetch works and I have data from the API in my database from earlier, however now the api is down (since 2-3 days) so I couldn't get a screenshot from the actual Heroku website. Proof the API is down on next image.

Checklist Items (1)

#4 Show an example of successful creation message





Proof API is Down.

Checklist Items (0)

COLLAPSE

Task #2 - Points: 1

Text: Screenshots of creation page code

Checklist

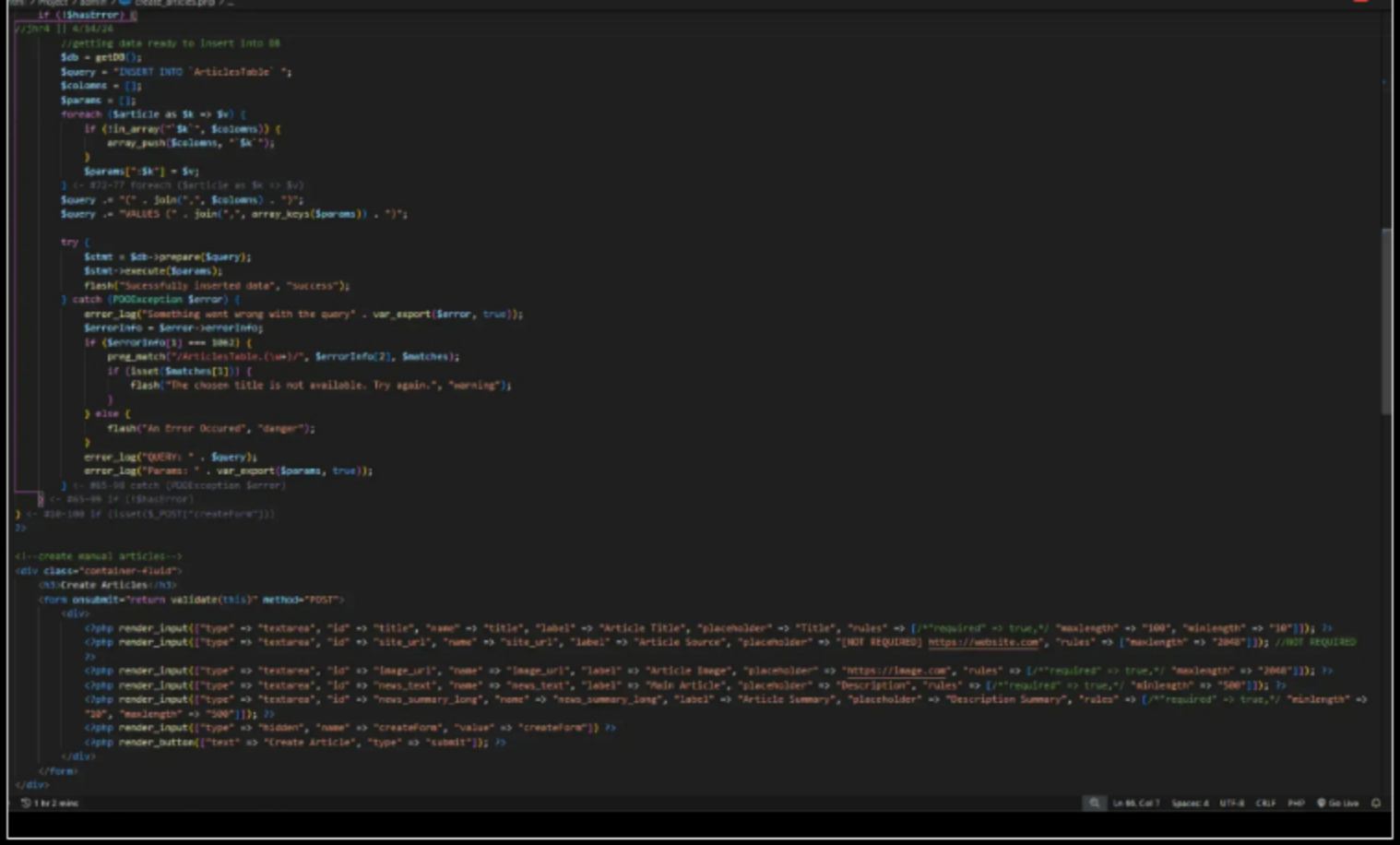
*The checkboxes are for your own tracking

#	Points	Details
#1	1	Form should have correct data types for each property being requested
#2	1	Form should have correct validation for each field (HTML, JS, and PHP)
#3	1	Successful creation should have a user-friendly message
#4	1	Any errors should have user-friendly messages
#5	1	Include the form/process for fetching API data
#6	1	Include some indicator between custom data and API data
#7	1	Include any other rules like role guards and login checks
#8	1	Include ucid/date comments for each code screenshot
#9	1	Clearly caption screenshots

Task Screenshots:

Gallery Style: Large View

Small Medium Large



Shows logic for manual creation page. Shows H1 title as an indicator that it's manual create article. The data types request are also correct (text for everything as it's all just text for article). Shows HTML validation Length check and existence (required). Line 84 shows creation successful message Line 86-94 Includes User Friendly messages on errors UCID and Date on top.

Checklist Items (7)

#1 Form should have correct data types for each property being requested

#2 Form should have correct validation for each field (HTML, JS, and PHP)

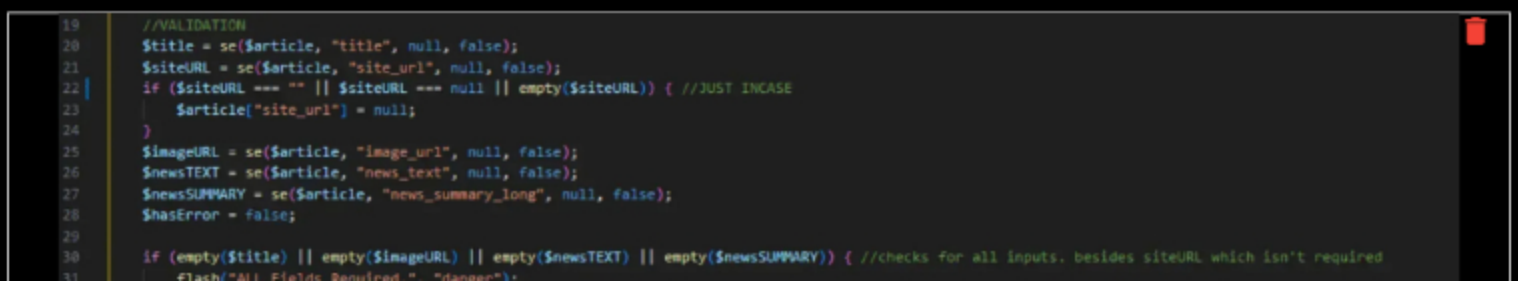
#3 Successful creation should have a user-friendly message

#4 Any errors should have user-friendly messages

#6 Include some indicator between custom data and API data

#8 Include ucid/date comments for each code screenshot

#9 Clearly caption screenshots



```php
        $hasError = true;
    }
    if (!empty($siteURL) && !preg_match('/(https?:\/\/)?(www\.)?[-a-zA-Z0-9@:%._\+~#=]{2,256}\.[a-z]{2,6}\b([-a-zA-Z0-9@:%_\+.~#?&\/\/=]*)/', $siteURL)) {
        flash("Invalid Source Link.", "danger");
        $hasError = true;
    }
    if (strlen($siteURL) >= 2048) {
        flash("Source Link Must Be Shorter Than 2048 Characters.", "danger");
        $hasError = true;
    }
    if (strlen($title) >= 100 || strlen($title) <= 10) {
        flash("Title Must 10-100 Characters.", "danger");
        $hasError = true;
    }
    if (!preg_match('/(https?:\/\/)?(www\.)?[-a-zA-Z0-9@:%._\+~#=]{2,256}\.[a-z]{2,6}\b([-a-zA-Z0-9@:%_\+.~#?&\/\/=]*)/', $imageURL)) {
        flash("Invalid Image Link.", "danger");
        $hasError = true;
    }
    if (strlen($imageURL) >= 2048) {
        flash("Image Link Must Be Shorter Than 2048 Characters.", "danger");
        $hasError = true;
    }
    if (strlen($newsTEXT) <= 500) {
        flash("News Body Must be 500 Characters or Greater.", "danger");
        $hasError = true;
    }
    if (strlen($newsSUMMARY) >= 500 || strlen($newsSUMMARY) <= 10) {
        flash("News Summary Must be 10-500 Characters.", "danger");
        $hasError = true;
    }
//jhr4 || 4/14/24
```

Manual Creation, PHP Validation. UCID date on bottom.

Checklist Items (1)

#2 Form should have correct validation for each field (HTML, JS, and PHP)

```
public_html > Project > admin > create_articles.php > script

<script>//jhr4 || 4/14/24
    function validate(form) {
        let title = form.title.value;
        let siteURL = form.site_url.value;
        let imageURL = form.image_url.value;
        let newsTEXT = form.news_text.value;
        let newsSUMMARY = form.news_summary_long.value;

        let isValid = true;

        //EXISTENCE of Everything besides siteURL which isn't required
        if (title === "" || imageURL === "" || newsTEXT === "" || newsSUMMARY === "") {
            flash("[Client] All fields be provided.", "danger");
            isValid = false;
        }
        //TITLE
        if (title.length <= 10 || title.length >= 100) {
            flash("[Client] Title Must Be 10-100 Characters.", "danger");
            isValid = false;
        }
        //siteURL
        if (siteURL !== "" && !/(https?:\/\/)?(www\.)?[-a-zA-Z0-9@:%._\+~#=]{2,256}\.[a-z]{2,6}\b([-a-zA-Z0-9@:%_\+.~#?&\/\/=]*)/.test(siteURL)) {
            flash("[Client] Invalid Source Link.", "danger");
            isValid = false;
        }
        if (siteURL.length >= 2048) {
            flash("[Client] Source Link Must Be Shorter Than 2048 Characters.", "danger");
            isValid = false;
        }
        //imageURL
        if (imageURL !== "" && !/(https?:\/\/)?(www\.)?[-a-zA-Z0-9@:%._\+~#=]{2,256}\.[a-z]{2,6}\b([-a-zA-Z0-9@:%_\+.~#?&\/\/=]*)/.test(imageURL)) {
            flash("[Client] Invalid Image Link.", "danger");
            isValid = false;
        }
        if (imageURL.length >= 2048) {
            flash("[Client] Image Must Be Shorter Than 2048 Characters.", "danger");
            isValid = false;
        }
        //newsText
        if (newsTEXT.length <= 500) {
            flash("[Client] News Body Must Be 500 Characters or Greater.", "danger");
            isValid = false;
        }
        //newsSUMMARY
        if (newsSUMMARY.length <= 10 || newsSUMMARY.length >= 500) {
            flash("[Client] News Summary Must be 10-500 Characters.", "danger");
            isValid = false;
        }

        return isValid;
    } <= 4121-170 function validate(form)
</script>
```

Manual creation page JS Validation UCID date on top.

Checklist Items (1)

#2 Form should have correct validation for each field (HTML, JS, and PHP)

public_html > Project > admin > create_articles.php > ...

```php
<?php
//note we need to go up 1 more directory
require(__DIR__ . "/../../../partials/nav.php");

if (!has_role("Admin")) {
    flash("You don't have permission to view this page", "warning");
    die(header("Location: " . get_url("home.php")));
}//jhr4 || 4/14/24

```

Role check on manual create page. UCID date on bottom.

Checklist Items (1)

#7 Include any other rules like role guards and login checks

```php
<?php //jhr4 4/14/24
//note we need to go up 1 more directory
require(__DIR__ . "/../../../partials/nav.php");

if (!has_role("Admin")) {
    flash("You don't have permission to view this page", "warning");
    die(header("Location: " . get_url("home.php")));
}
?>
<?php
if(isset($_POST["articleDays"])){
    //validation
    $days = se($_POST, "articleDays", null, false);
    $hasError = false;

    if ($days === null || $days === ""){
        flash("ALL Fields Required.", "danger");
        $hasError = true;
    }
    if (!($days > 0)){
        flash("Days value must be greater than 0.", "danger");
        $hasError = true;
    }

    if (!$hasError) {
        $result = get('https://spacenews.p.rapidapi.com/datenews/'.$_POST["articleDays"], "SPACE_API_KEY", $data = ["days" => $_POST["articleDays"]], true, 'spacenews.p.rapidapi.com');

        error_log("API Response: " . var_export($result, true));
        if (se($result, "status", 400, false) == 200 && isset($result["response"])) {
            $result = json_decode($result["response"], true);
        } else {
            $result = [];
        }

        //date transformation
        $data= [];
        foreach ($result as $article){
            $article["api_id"] = $article["id"];
            unset($article["id"]);
            $article["api_timestamp"] = $article["timestamp"];
            unset($article["timestamp"]);
            $temp["api_timestamp"] = str_replace("T", " ", $article["api_timestamp"]);
            $temp["api_timestamp"] = str_replace("Z", "", $temp["api_timestamp"]);
            $article["api_timestamp"] = $temp["api_timestamp"];
            unset($article["news_summary_short"]);
            unset($article["hashtags"]);
            array_push($data, $article);
        } <- #37-46 foreach ($result as $article)
        $result = $data;

        //getting data ready to insert into DB
        $db = getDB();
        $query = "INSERT INTO `ArticlesTable` ";

        //setting up columns
```

API Fetch Code. Role Gard Admin Check on top. PHP Validation lines 12-23 Code for API Fetching (Data transformation on bottom) UCID and Date on top.

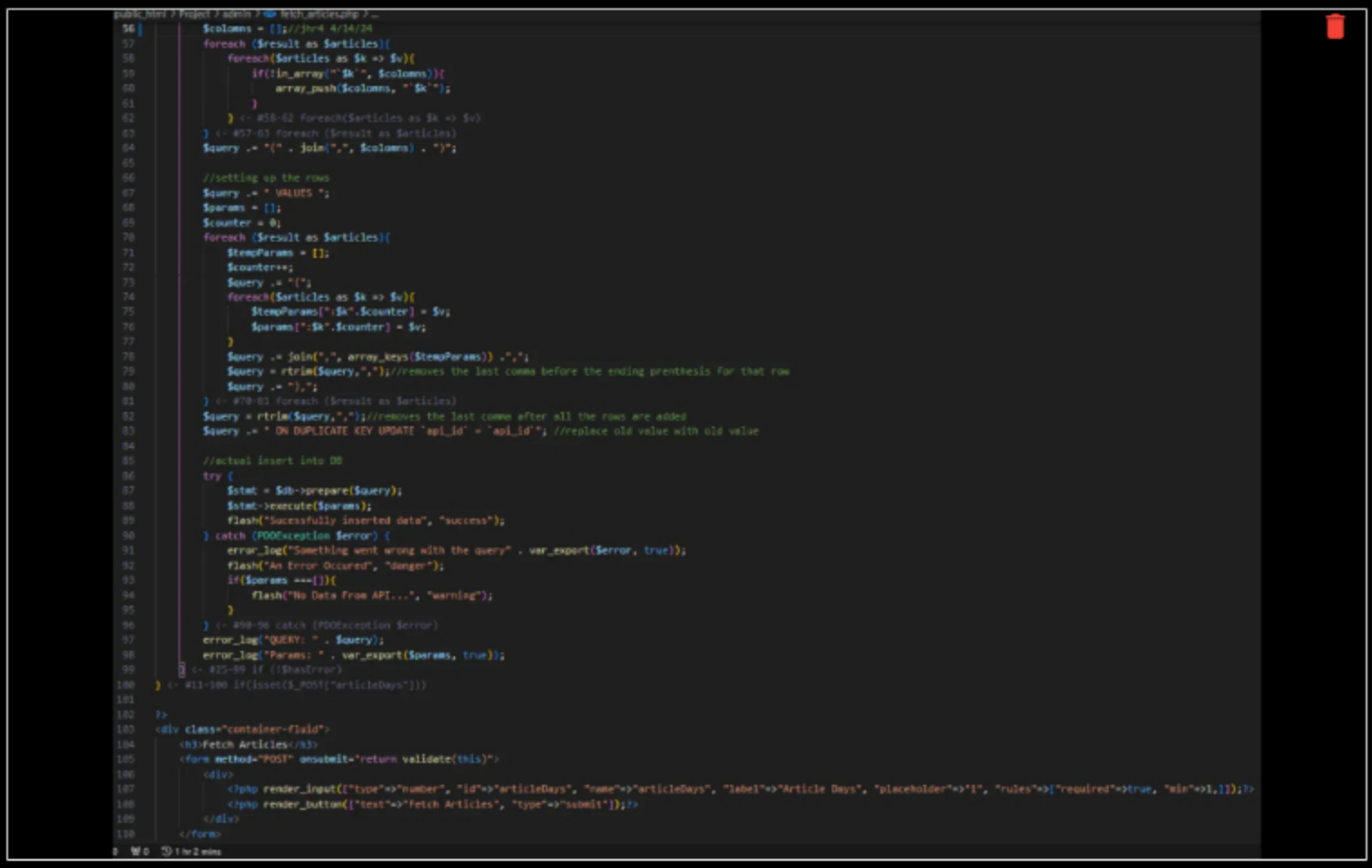
Checklist Items (4)

#2 Form should have correct validation for each field (HTML, JS, and PHP)

#5 Include the form/process for fetching API data

#7 Include any other rules like role guards and login checks

#8 Include ucid/date comments for each code screenshot



```
$colomns = [];//jhr4 4/14/24
foreach ($result as $articles){
    foreach($articles as $k => $v){
        if(!in_array("`$k`", $colomns)){
            array_push($colomns, "`$k`");
        }
    } <- #58-62 foreach($articles as $k => $v)
} <- #57-63 foreach ($result as $articles)
$query .= "(" . join(",", $colomns) . ")";

//setting up the rows
$query .= " VALUES ";
$params = [];
$counter = 0;
foreach ($result as $articles){
    $tempParams = [];
    $counter++;
    $query .= "(";
    foreach($articles as $k => $v){
        $tempParams[":$k".$counter] = $v;
        $params[":$k".$counter] = $v;
    }
    $query .= join(",", array_keys($tempParams)) .",";
    $query = rtrim($query,",");//removes the last comma before the ending prenthesis for that row
    $query .= "),";
} <- #70-81 foreach ($result as $articles)
$query = rtrim($query,",");//removes the last comma after all the rows are added
$query .= " ON DUPLICATE KEY UPDATE `api_id` = `api_id`"; //replace old value with old value

//actual insert into DB
try {
    $stmt = $db->prepare($query);
    $stmt->execute($params);
    flash("Sucessfully inserted data", "success");
} catch (PDOException $error) {
    error_log("Something went wrong with the query" . var_export($error, true));
    flash("An Error Occured", "danger");
    if($params ===[]){
        flash("No Data From API...", "warning");
    }
} <- #90-96 catch (PDOException $error)
error_log("QUERY: " . $query);
error_log("Params: " . var_export($params, true));
} <- #25-99 if (!$hasError)
} <- #11-100 if(isset($_POST["articleDays"]))

?>
<div class="container-fluid">
    <h3>Fetch Articles</h3>
    <form method="POST" onsubmit="return validate(this)">
        <div>
            <?php render_input(["type"=>"number", "id"=>"articleDays", "name"=>"articleDays", "label"=>"Article Days", "placeholder"=>"1", "rules"=>["required"=>true, "min"=>1,11]);?>
            <?php render_button(["text"=>"Fetch Articles", "type"=>"submit"]);?>
        </div>
    </form>
```

Shows API Data process for inserting into DB . Line 91-94 Shows Error messages code. Line 89 shows Success message code. Line 104 H3 tag "Fetch Article" indicates its API Data. Also when getting added into the DB the API_ID col is filled for api data. Data Types is int and the type of form is number too. Line 107 HTML Validation: required and minimum number = 1 UCID and date on top.

Checklist Items (5)

#1 Form should have correct data types for each property being requested

#2 Form should have correct validation for each field (HTML, JS, and PHP)

#3 Successful creation should have a user-friendly message

#4 Any errors should have user-friendly messages

#5 Include the form/process for fetching API data

```
<script>//jhr4 4/14/24
function validate(form) {
    let days = form.articleDays.value;
    let isValid = true;

    //EXISTANCE of Everything
    if (days === "" || articleDays === null){
        flash("[Client] Days feild must be provided.", "danger");
        isValid = false;
    }
    //validation of days
    if (!(days > 0)){
        flash("[Client] Days must be greater than 0.", "danger");
        isValid = false;
    }
    //return isValid;
} <- #113-128 function validate(form)

</script>
```

API Creation JS Validation. UCID Date on top.

Checklist Items (1)

#2 Form should have correct validation for each field (HTML, JS, and PHP)

COLLAPSE

Task #3 - Points: 1

Text: Screenshot of records from DB

Checklist

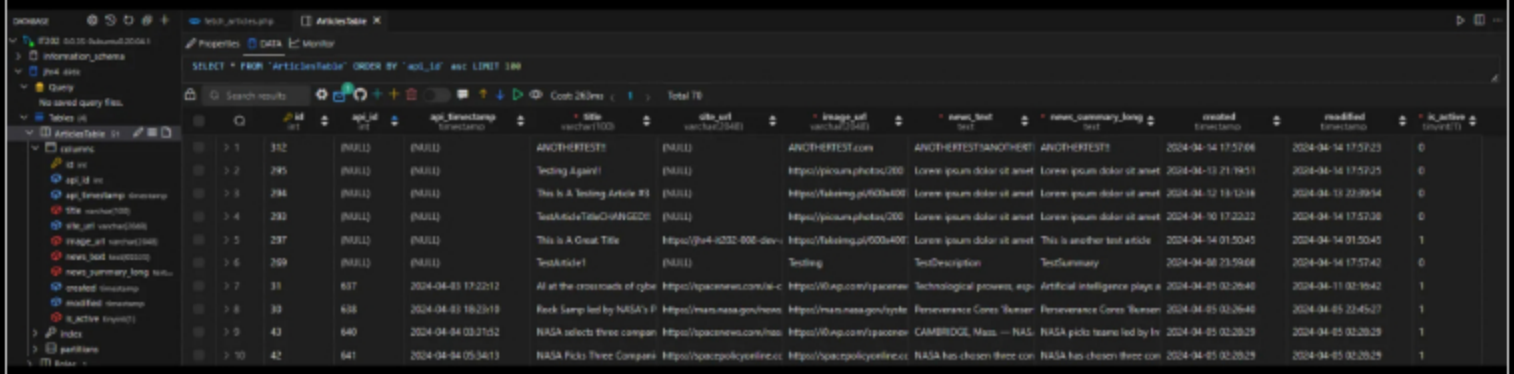
*The checkboxes are for your own tracking

#	Points	Details
#1	1	Show at least one record fetched from the API
#2	1	Show at least one record created via the creation form
#3	1	Note what differs
#4	1	Clearly caption screenshots

Task Screenshots:

Gallery Style: Large View

Small Medium Large



This screenshot shows API fetched articles, records 7 (id 31) and below. This is identified by the api_id not being null. The screenshot shows manually created articles, records 6 (id 269) and above. This is identified by the api_id being null.

Checklist Items (4)

#1 Show at least one record fetched from the API

#2 Show at least one record created via the creation form

#3 Note what differs

#4 Clearly caption screenshots

COLLAPSE

Task #4 - Points: 1

Text: Explain the process

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Provide a high-level step-by-step of how fetching API data works and gets added to your DB (include how duplicates are handled)
#2	1	Provide a high-level step-by-step of how creating custom data works and gets added to your DB (include how duplicates are handled)
#3	1	Briefly describe the validations for the applicable fields
#4	1	Describe how duplicate data is handled

Response:

Fetching API Data:
For this page first the user enters a number of days to get the articles over those days and submits the form. After this the API process occurs on the server side. Data is fetched by connecting to the API with curl and sending a request to the API with the key and data (# of days) the user wants. After this the response data is gotten successfully the result is in json so it is decoded into an associative array. After this the data is converted based on what's needed and the format I want. In this case I unset some things like hashtags and the short summary that came with the response. Furthermore, I had to change the date format to something that works with the database.
After this, the data is looped through to prepare the query and params. For in the loop each articles data is taken as keys value pairs. The keys were taken to create the columns of the query request. The values were used for the actual data to create placeholders in the query and the actual values in the params. After this the database is called and the the query is prepared for the database and the params are executed.
Upon execution if there's duplicate api_id's it means the article already exists and the article just is updated with its self (there should be no change basically). Upon execution there's a success message shown or a error if it occurs.

Manual Create:
For the manual create there's a similar processes without the API part. The user enters in the values on the form on the website page and this is validated by HTML first then JS. Upon submission it's also validated on the server via PHP. For each of the validation the existence of data is checked, and the format (length for most of them). For the article and image url data regex is also used in PHP and JS to verify if it's in a actual link format. If there's an error a message is shown. Else the this data goes onto the next stage. After this the query is prepared by making the query and inserting the actual values into the params which will relate to the placeholders. If there is no duplicate or error a successful message is shown and the data is added to the db. Else a error message is shown. To make sure the article doesn't already exist in the db the code tries to update the data, but if the title is already existing there will be an error as there's an unique constraint. If it meets the error code of 1062 it will display the message article title in use already.

COLLAPSE

Task #5 - Points: 1

Text: Add related links

Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Include the heroku prod link for this page
#2	1	Add the pull request link for the branch related to this feature Note: the link should end with /pull/#. Same pull request shouldn't be used for each feature

URL #1
https://github.com/Jhr-4/jhr4-IT202-008/pull/55

URL #2
https://github.com/Jhr-4/jhr4-IT202-008/pull/57

URL #3
https://github.com/Jhr-4/jhr4-IT202-008/pull/59

URL #4
https://jhr4-it202-008-prod-af7c79552123.herokuapp.com/Project/admin/create_role.php

URL #5
https://jhr4-it202-008-prod-af7c79552123.herokuapp.com/Project/admin/fetch_articles.php

Data List Page (many entities) (2 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshots of the list page

Checklist

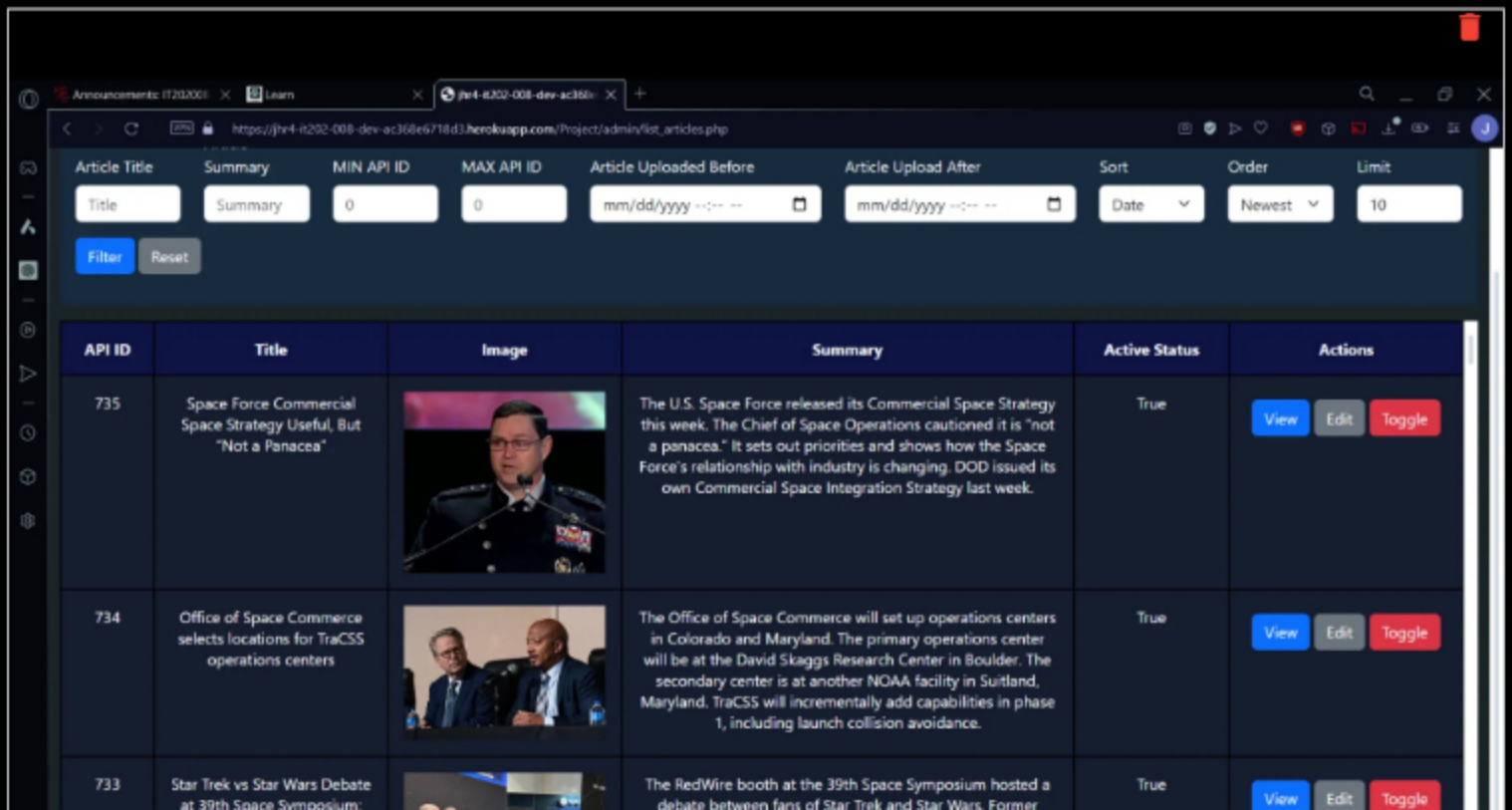
*The checkboxes are for your own tracking

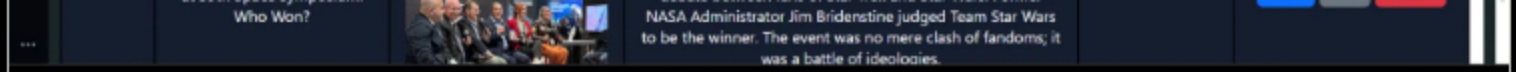
#	Points	Details
#1	1	Show the page of your entities listed (have a reasonable number shown)
#2	1	Show the filter/sort form based on your data and the required limit field
#3	1	Demonstrate a few varied filters/sorts
#4	1	Demonstrate a filter that doesn't have any records (should show an appropriate message)
#5	1	Each list item should have a link of single view (i.e., details), edit, and delete (some of which may only be visible to admin users)
#6	1	Each list item should have a summary of the entity (likely won't be the entire entity data)
#7	1	Design/Style should be considered (i.e., bootstrap, custom css, etc)
#8	1	Make sure the heroku dev url is visible in the address bar
#9	1	Clearly caption screenshots

Task Screenshots:

Gallery Style: Large View

Small Medium Large





No filter with entities listed. There's a summary of the article. Has a view button, edit button, and delete (toggle) button. There is a clean design via a tables of Bootstraps and it's easy to read/use. Heroku dev link in URL

Checklist Items (6)

#1 Show the page of your entities listed (have a reasonable number shown)

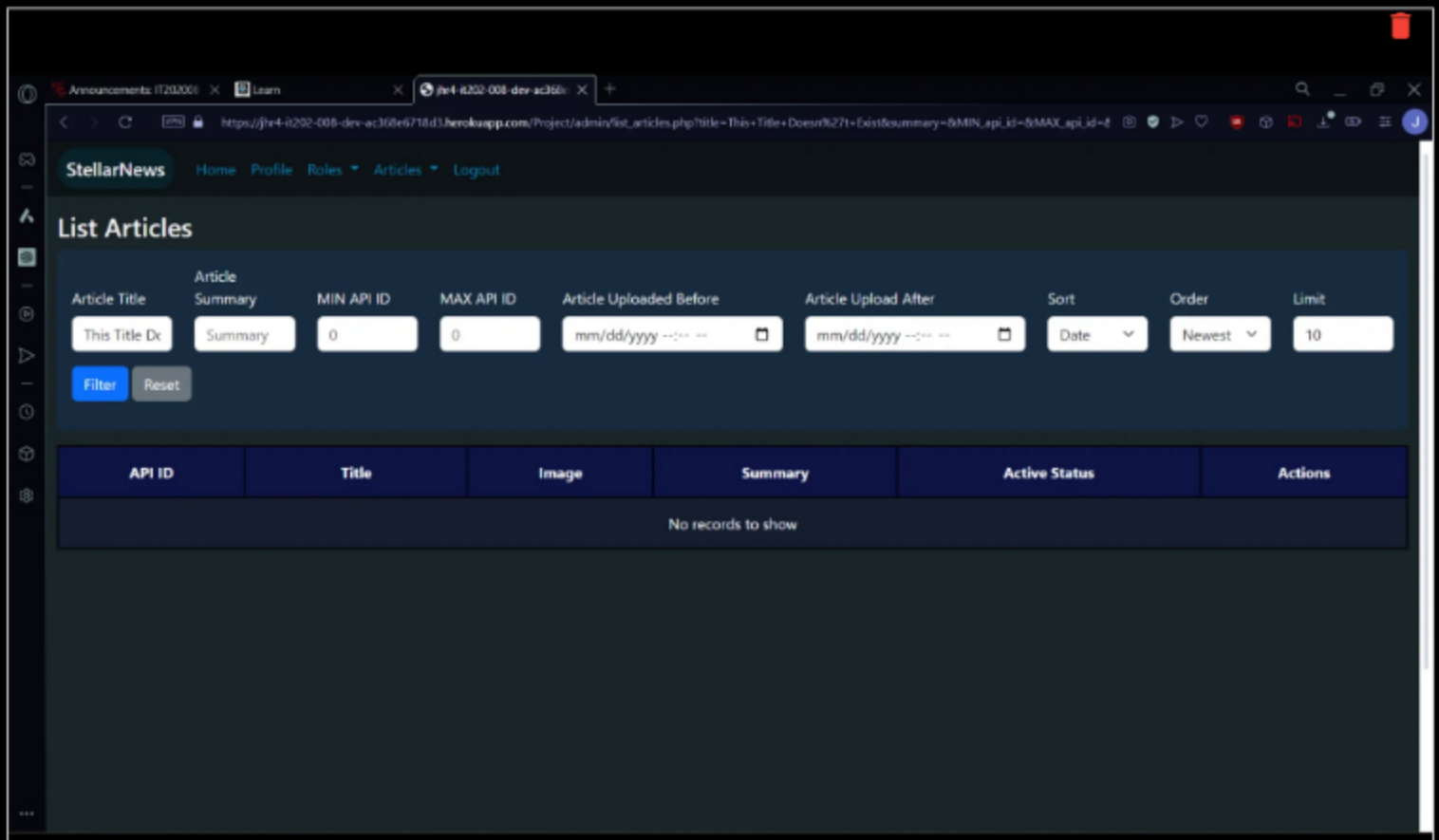
#5 Each list item should have a link of single view (i.e., details), edit, and delete (some of which may only be visible to admin users)

#6 Each list item should have a summary of the entity (likely won't be the entire entity data)

#7 Design/Style should be considered (i.e., bootstrap, custom css, etc)

#8 Make sure the heroku dev url is visible in the address bar

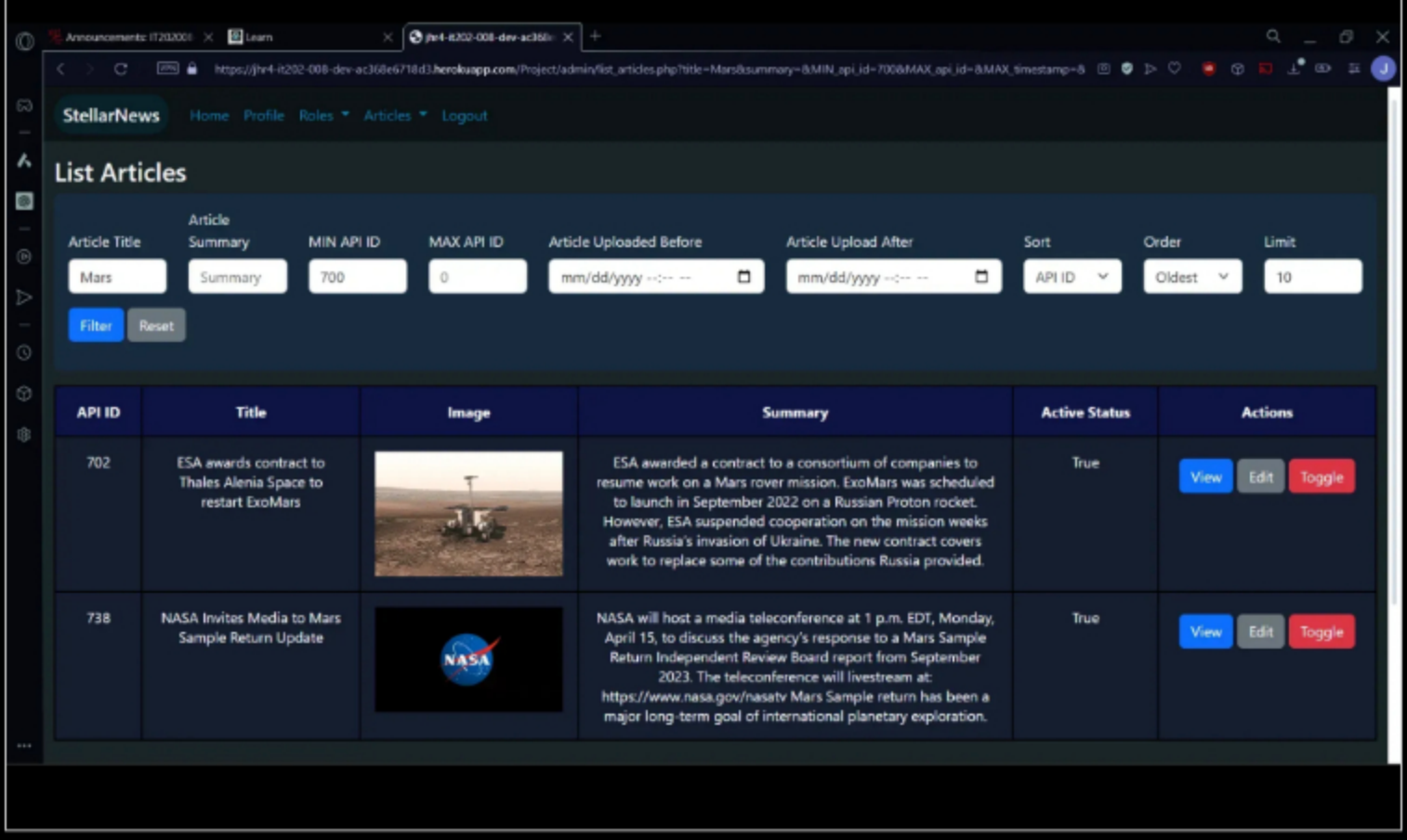
#9 Clearly caption screenshots



Filter of "This Title Doesn't Exist," Showing the text of "no records to show" in the table.

Checklist Items (1)

#4 Demonstrate a filter that doesn't have any records (should show an appropriate message)

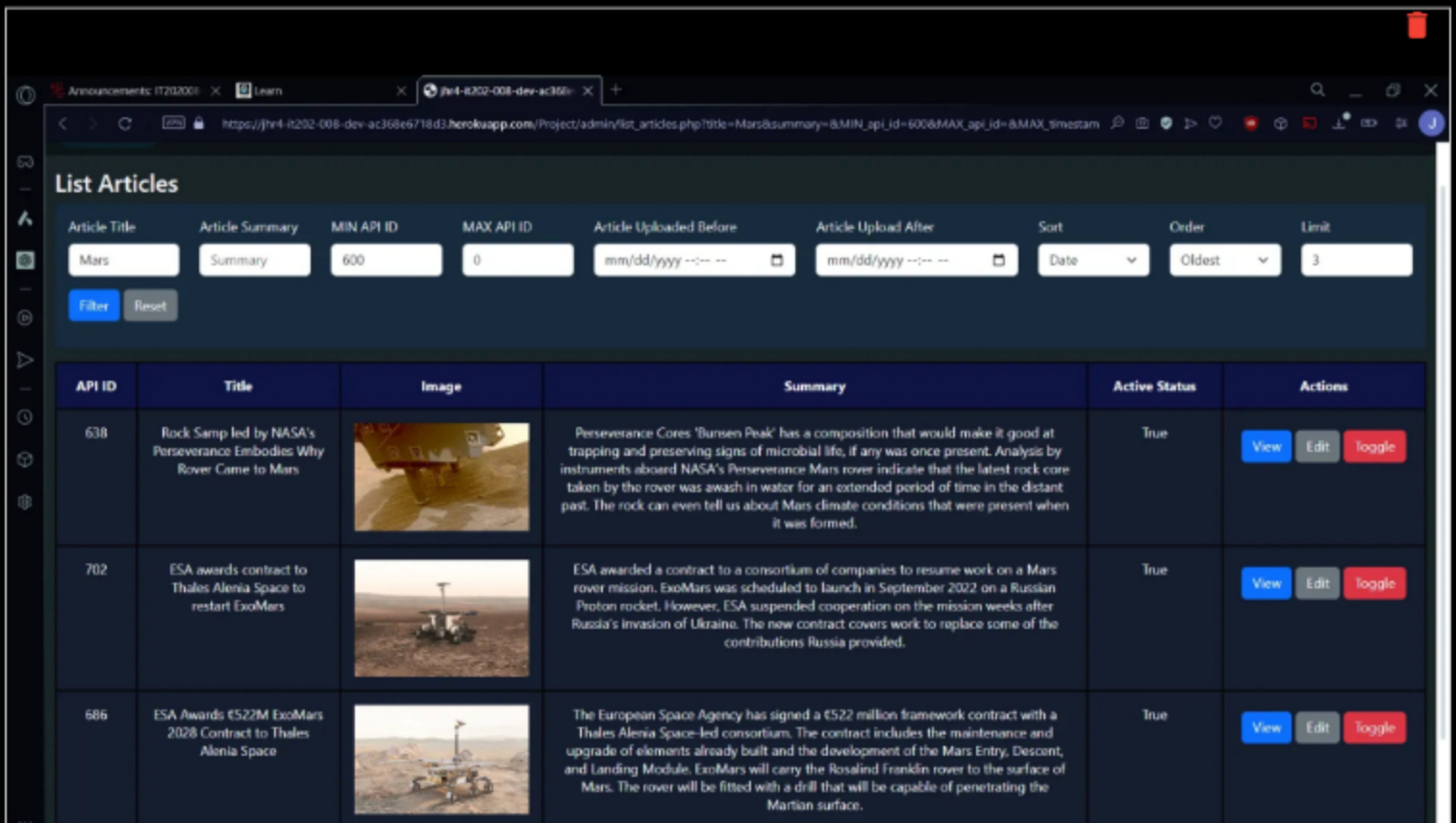


Filter of title "Mars" and MAX ID of 700 sorted by API ID, Oldest first, and limit of 10 (but there were just 2 entities in this case).

Checklist Items (2)

#2 Show the filter/sort form based on your data and the required limit field

#3 Demonstrate a few varied filters/sorts



Filter of title "Mars" and MAX ID of 600 sorted by Date, Oldest first, and limit of 3.

Checklist Items (2)

#2 Show the filter/sort form based on your data and the required limit field

#3 Demonstrate a few varied filters/sorts

COLLAPSE

Task #2 - Points: 1

Text: Screenshots of the list page code

Checklist

*The checkboxes are for your own tracking

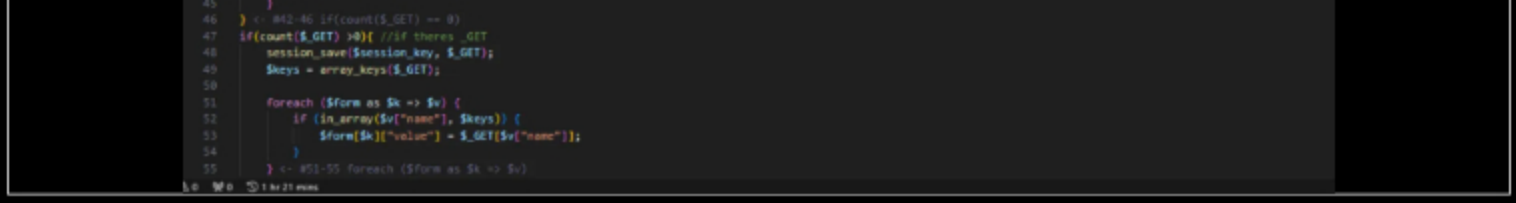
#	Points	Details
#1	1	Show the filter/sort form generation
#2	1	Show the DB query and how the filter/sort is handled (including the restriction on the limit field)
#3	1	Show how the output is generated and displayed
#4	1	Show any restrictions like role guard or login checks
#5	1	Include ucid/date comments for each code screenshot
#6	1	Clearly caption screenshots

Task Screenshots:

Gallery Style: Large View

Small Medium Large

```php
<?php //jhr4 4/14/24
//note we need to go up 1 more directory
require(__DIR__ . "/../../../partials/nav.php");

if (!has_role("Admin")) {
    flash("You don't have permission to view this page", "warning");
    die(header("Location: " . get_url("home.php")));
}

$form = [
    ["type"=>"input", "id"=>"title", "name"=>"title", "label"=>"Article Title", "placeholder"=>"Title", "include_margin" => false],
    ["type"=>"input", "id"=>"summary", "name"=>"summary", "label"=>"Article Summary", "placeholder"=>"Summary", "include_margin" => false],

    ["type"=>"number", "id"=>"MIN_api_id", "name"=>"MIN_api_id", "label"=>"MIN API ID", "placeholder"=>"0", "include_margin" => false],
    ["type"=>"number", "id"=>"MAX_api_id", "name"=>"MAX_api_id", "label"=>"MAX API ID", "placeholder"=>"0", "include_margin" => false],

    ["type"=>"datetime-local", "id"=>"MAX_timestamp", "name"=>"MAX_timestamp", "label"=>"Article Uploaded Before", "include_margin" => false],
    ["type"=>"datetime-local", "id"=>"MIN_timestamp", "name"=>"MIN_timestamp", "label"=>"Article Upload After", "include_margin" => false],

    ["type" => "select", "name" => "sort", "label" => "Sort", "options" => ["created" => "Date", "api_id" => "API ID"], "include_margin" => false],
    ["type" => "select", "class"=>"test","name" => "order", "label" => "Order", "options" => ["desc" => "Newest", "asc" => "Oldest"], "include_margin" => false],

    ["type" => "number", "name" => "limit", "label" => "Limit", "value" => "10", "placeholder"=>"10", "include_margin" => false],
]; <- #10-24 $form =
//LOADING  TABLE/ARTICLES

$query = "SELECT id, api_id, title, site_url, image_url, news_text, news_summary_long, is_active FROM  `ArticlesTable` WHERE 1=1";
$params = [];
$session_key = $_SERVER["SCRIPT_NAME"];

//RESET/CLEAR FILTERS BUTTON
$is_clear = isset($_GET["clear"]);
if ($is_clear) {
    session_delete($session_key);
    unset($_GET["clear"]);
    die(header("Location: " . $session_key));
} else {
    $session_data = session_load($session_key);
}

//FILTERING/SORTING FROM _GET
if(count($_GET) == 0){ //if doesnt exist
    if($session_data){
        $_GET = $session_data;
```



Shows Role Admin check on top. Shows Filter/Sort form creation on top. Bottom/Middle shows start of DB query being created and clearing filters when the button is clicked by cleaning the session too. UCID and Date on Top.

Checklist Items (5)

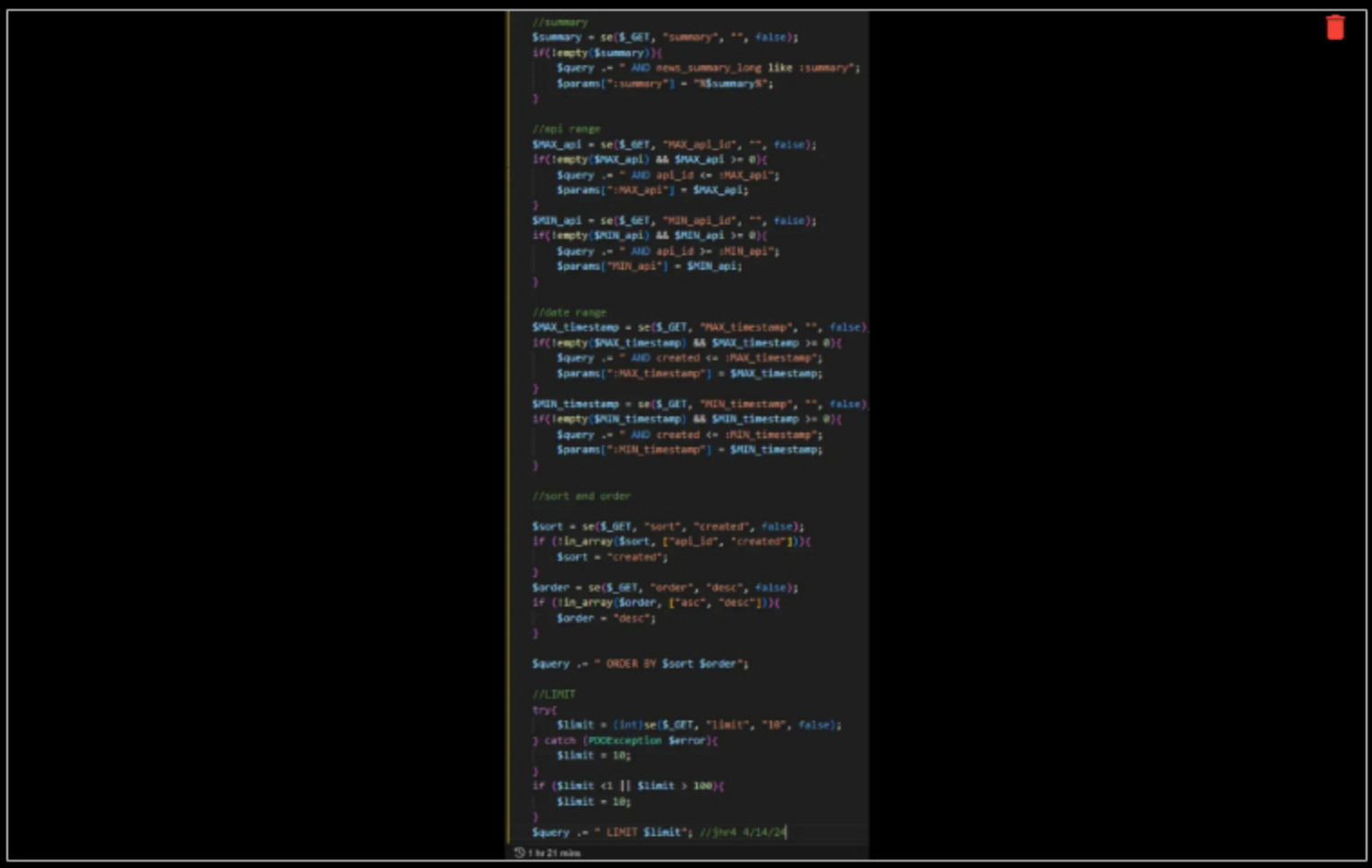
#1 Show the filter/sort form generation

#2 Show the DB query and how the filter/sort is handled (including the restriction on the limit field)

#4 Show any restrictions like role guard or login checks

#5 Include ucid/date comments for each code screenshot

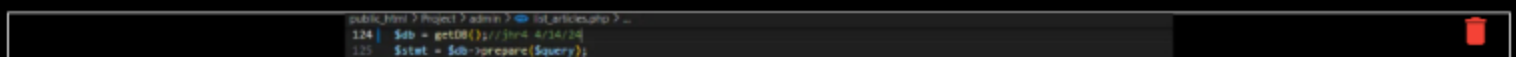
#6 Clearly caption screenshots



Shows more of the query being build for the filters and sorting. Shows restrictions on limit filed. If smaller than 1 or greater than 100 limit is changed to 10. UCID and Date on Bottom.

Checklist Items (1)

#2 Show the DB query and how the filter/sort is handled (including the restriction on the limit field)



```php
$results = [];
try{
    $stmt -> execute($params);
    $r = $stmt->fetchAll();
    if($r){
        $results = $r;
    }
} catch(PDOException $error) {
    error_log("Error fetching stocks: " . var_export($error, true));
    flash("An error occured", "danger");

} <-- #133-137 catch(PDOException $error)
for($i=0; $i<count($results); $i++){ //adds data for null values (that were added by manual create)
    foreach($results[$i] as $k => $v){
        if ($v === null){
            $results[$i][$k] = "N/A";
        }
        if ($k ==='is_active' && $v===1){
            $results[$i][$k] = "True";
        } else if ($k ==='is_active' && $v===0){
            $results[$i][$k] = "False";
        }
    } <-- #139-148 foreach($results[$i] as $k => $v)
} <-- #138-149 for($i=0; $i<count($results); $i++)


$table = ["data" => $results, "extra_classes" => "listTable", "ignored_columns" => ["id", "news_text", "site_url"],
          "edit_url"=>get_url("admin/edit_articles.php"),
          "delete_url"=>get_url("admin/delete_articles.php"), "delete_label"=>"Toggle",
          "view_url"=>get_url("view_articles.php"),
          "header_override"=>["API ID", "Title", "Image", "Summary", "Active Status"]
        ]; <-- #152-157 $table =
?>

<div class="container-fluid">
    <h3>List Articles</h3>
    <div class="card card-body border-0 bg-primary bg-opacity-10 mb-3">
        <form method="GET">
            <div class="row mb-3" style="align-items: baseline;">
            <?php foreach($form as $k => $v) :?>
                <div class="col">
                    <?php render_input($v); ?>
                </div>
            <?php endforeach; ?>
            </div>
            <?php render_button(["text"=>"Filter", "type"=>"submit"]);?>
            <a href="?clear" class="btn btn-secondary">Reset</a>
        </form>
    </div>
    <div class="tableDiv">
    <?php render_table($table);?>
    </div>
</div>
```

Shows how the output is generated via the render table function which just creates columns based on the keys from the database results and the values associated to it for the rows. UCID and Date on top. Shows actual creation/generation of filter/sort form on bottom with render input.

Checklist Items (3)

#1 Show the filter/sort form generation

#3 Show how the output is generated and displayed

#5 Include ucid/date comments for each code screenshot

^COLLAPSE^

Task #3 - Points: 1

Text: Explain how the page works

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Provide the high-level steps of how the filter/sorting works and how data is displayed
#2	1	Summarize what you're showing on the screen
#3	1	Mention your design/style choice
#4	1	Mention which users can interact with the view, edit, and delete links

Response:

Filter/Sort:

To filter and sort first the user fills out the form based on what they want to filter then this is sent as a get request (get rq so the user can share this filter link to someone else). When getting this data, each key of the data is checked if its empty or not. If there's a data for it the query and params are updated. For the filters with specific words the LIKE operator is used to find that word pattern. In this cause %input% would check if that character or word exists in that column in any spot. For the range filters, the column is checked if there's a value in the table that's bigger or smaller than the one passed through based on if the user put it for maximum or minimum. For the sorting options if no input is provided the articles are ORDER(ed) BY CREATED DESC which means they are in order of newest to oldest.
Furthermore, the limit is set to 10 if there's no limit or if the user tries to add a negative or 100+ limit on how many articles to show. After these params and query are prepared the db is called and it's executed selecting and fetching all the values and applying the constraints/sorting based on the filter/sorting inputted in the query. Upon error a message is shown.

Displaying:
To display these results the fetched results are looped through for each article as key value pairs. For values that are null it's updated to "N/A" which is easier for the user to understand and the is_active which is 0 or 1 is changed to true/false. After this a table is crated basically by looping through each article and adding the columns based on the keys and the values for them in each row. This is done through render_table() function in table.php which takes in columns/data to ignore which is done by matching this column to the key in the fetched result and just ignoring the column/keys and values for it. If the column to ignore is specified when looping through the fetched data, it's skipped over, thus not showing it. After all this the columns shown are: API ID, Title, Image, Summary, Active Status, and Actions. Thefore, the ignored columns include the id (primary column), news_text(the paragraphs of article), and the site url. API ID shows N/A if it's an article that was manually created. Furthermore in the Actions column actions buttons for convince are shown: View, Edit, and Delete which does what it states and will be discussed in that section.

The style is a simple table which is only accessible by the admin and they are the only ones that can interact with the buttons too as it's a locked page (no users can come to this list page without the admin role).
There however, is another list which shows cards view for the users in the same way as on this page. This is accessible by anyone logged in. This page has no buttons besides view which is accessible by everyone logged in too. Basically there are two list pages one for users and user action (viewing) and the admin list which is for admins and has easier to see data with buttons for convince to edit or delete or view.
Additionally in the users list page when looping over the data if it has been toggled/deleted the articles loop is skipped/continued therefore, not creating a card for it and not displaying it to the user. In the admin list page, the admin actually can see the deleted article.

COLLAPSE

Task #4 - Points: 1

Text: Add related links

Checklist

*The checkbox are for your own tracking

#	Points	Details
#1	1	Include the heroku prod link for this page
#2	1	Add the pull request link for the branch related to this feature Note: the link should end with /pull/#. Same pull request shouldn't be used for each feature

URL #1
https://github.com/Jhr-4/jhr4-IT202-008/pull/55

URL #2

URL #2
https://github.com/Jhr-4/jhr4-IT202-008/pull/58

URL #3
https://jhr4-it202-008-prod-af7c79552123.herokuapp.com/Project/admin/list_articles.php

URL #4
https://jhr4-it202-008-prod-af7c79552123.herokuapp.com/Project/home.php

View Details Page (single entity) (1 pt.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshots of the details page

Checklist

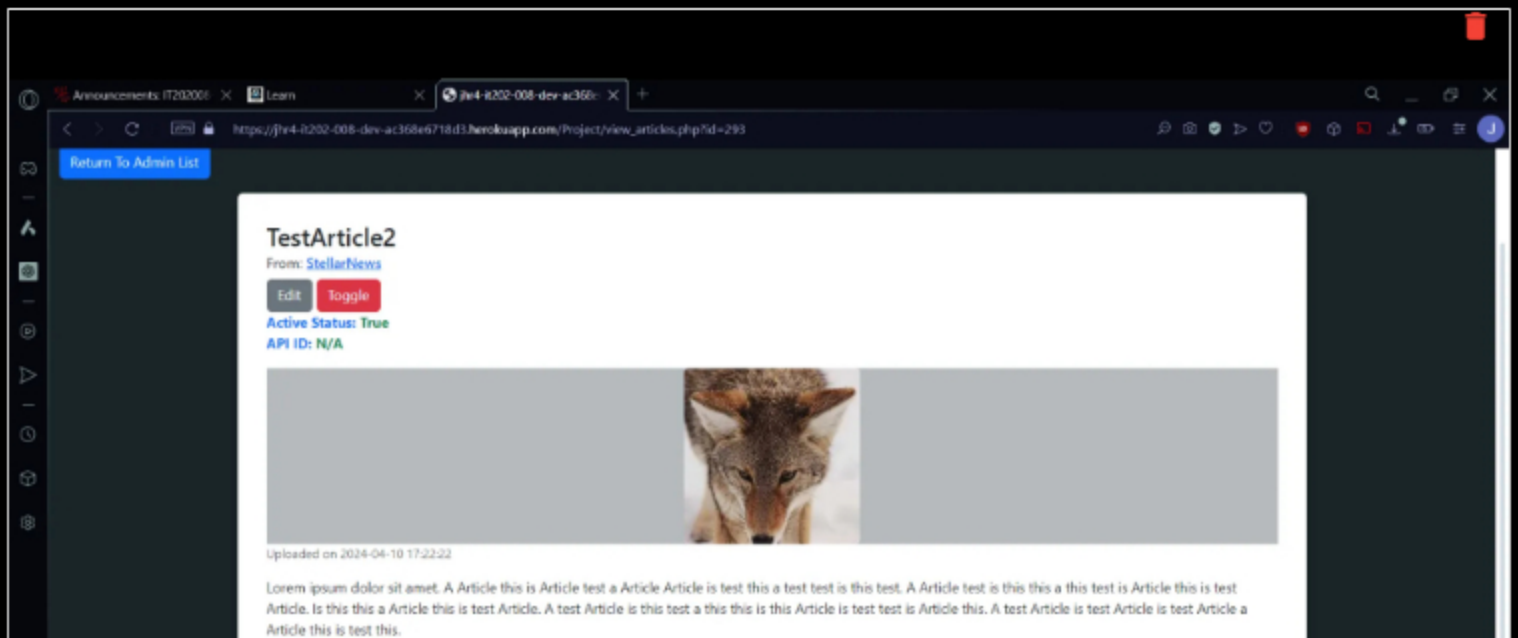
*The checkboxes are for your own tracking

#	Points	Details
#1	1	Entity should be fetch by id (via the url)
#2	1	A missing id should redirect back to the list page with an applicable message
#3	1	Design/Style should be considered (i.e., bootstrap, custom css, etc)
#4	1	Data shown should be more detailed/inclusive than the summary view
#5	1	There should be a link to edit the entity (this may be an admin-only thing, but it should be present for the respective role)
#6	1	There should be a link to delete the entity (this may be an admin-only thing, but it should be present for the respective role)
#7	1	Make sure the heroku dev url is visible in the address bar
#8	1	Clearly caption screenshots

Task Screenshots:

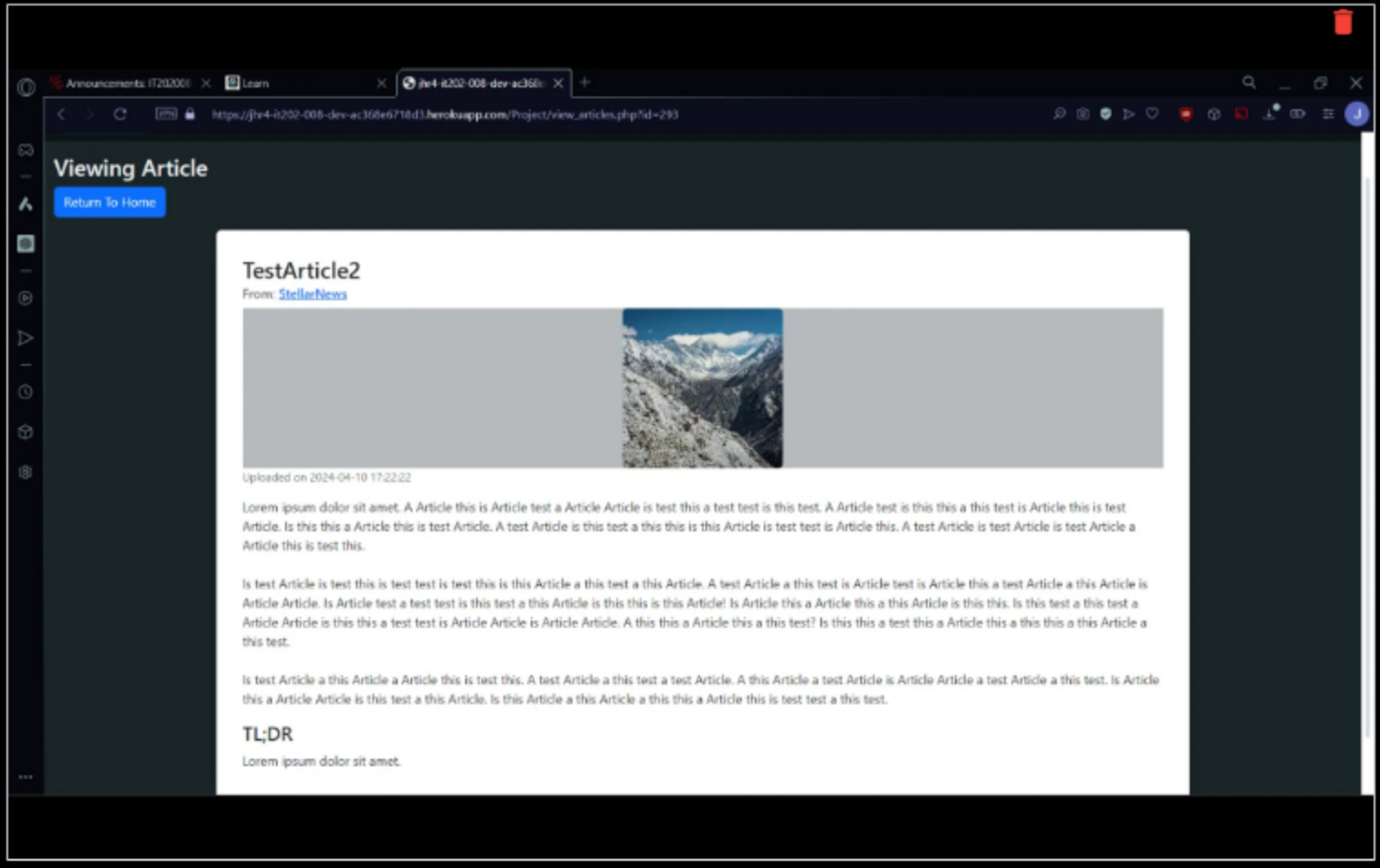
Gallery Style: Large View

Small Medium Large



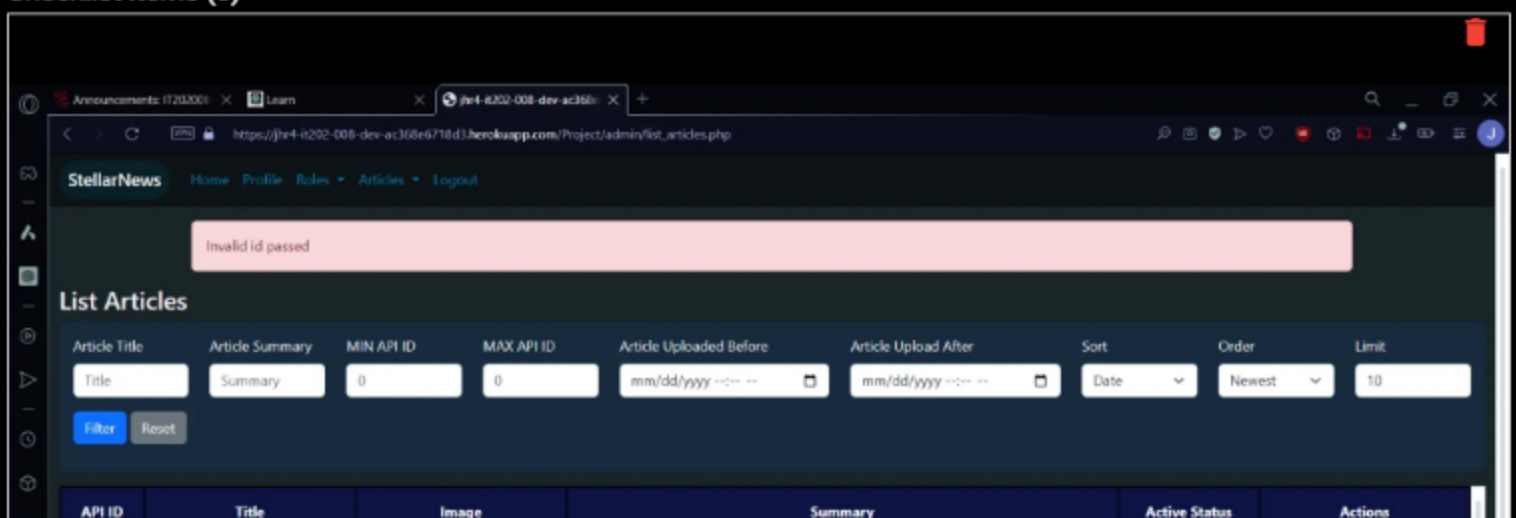
Admin View: Single Page View Fetch by ID in URL. Design using cards in Bootstrap. Data is more detailed than list page, Includes upload time/date and the actual body paragraphs text. For admin there's a link to delete (toggle), edit, and go back (return).

Checklist Items (0)



Non-Admin User View. No edit or delete button, and return (back) to home page button.

Checklist Items (0)



735	Space Force Commercial Space Strategy Useful, But "Not a Panacea"		The U.S. Space Force released its Commercial Space Strategy this week. The Chief of Space Operations cautioned it is "not a panacea." It sets out priorities and shows how the Space Force's relationship with industry is changing. DOD issued its own Commercial Space Integration Strategy last week.	True	View Edit Toggle
734	Office of Space Commerce selects locations for TraCSS operations centers		The Office of Space Commerce will set up operations centers in Colorado and Maryland. The primary operations center will be at the David Skaggs Research Center in Boulder. The secondary center is at another NOAA facility in Suitland, Maryland. TraCSS will incrementally add capabilities in phase 1, including launch collision avoidance.	True	View Edit Toggle

Missing ID goes back to list page with message on top.

Checklist Items (1)

#3 Design/Style should be considered (i.e., bootstrap, custom css, etc)

COLLAPSE

Task #2 - Points: 1

Text: Screenshots of the details page code

Checklist

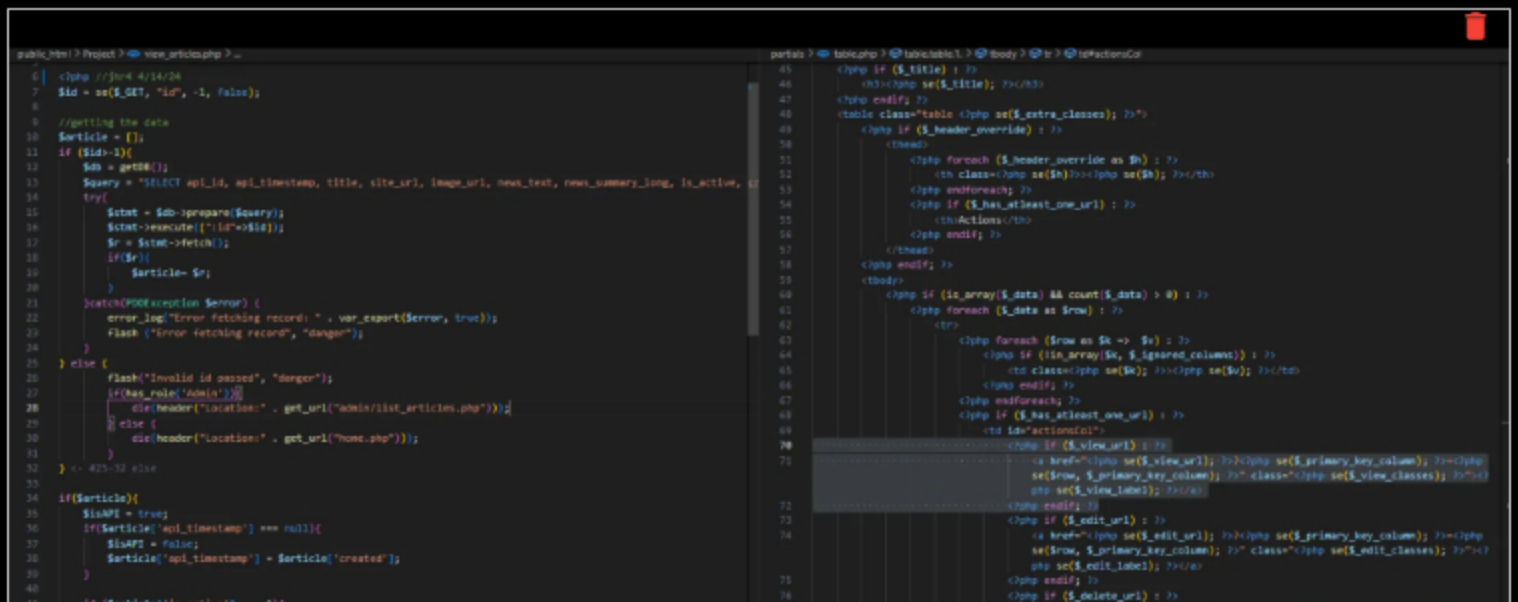
*The checkboxes are for your own tracking

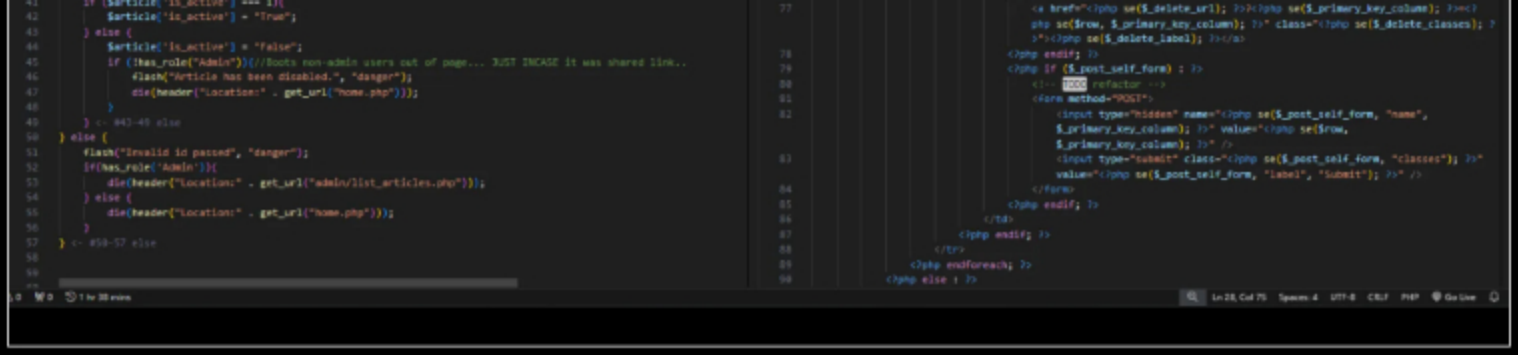
#	Points	Details
#1	1	Show how id is fetched
#2	1	Show the DB query to get the record
#3	1	Show the code related to presenting the data and showing the links
#4	1	Include ucid/date comments for each code screenshot
#5	1	Clearly caption screenshots

Task Screenshots:

Gallery Style: Large View

Small Medium Large





Shows how Id is fetched via the select query. The id is gotten from url/get request which is created via the button that's clicked as seen on the right which puts the primary column id as part of the href link. Shows how DB query gets the record by selecting the specific columns on top. UCID and Date on Top.

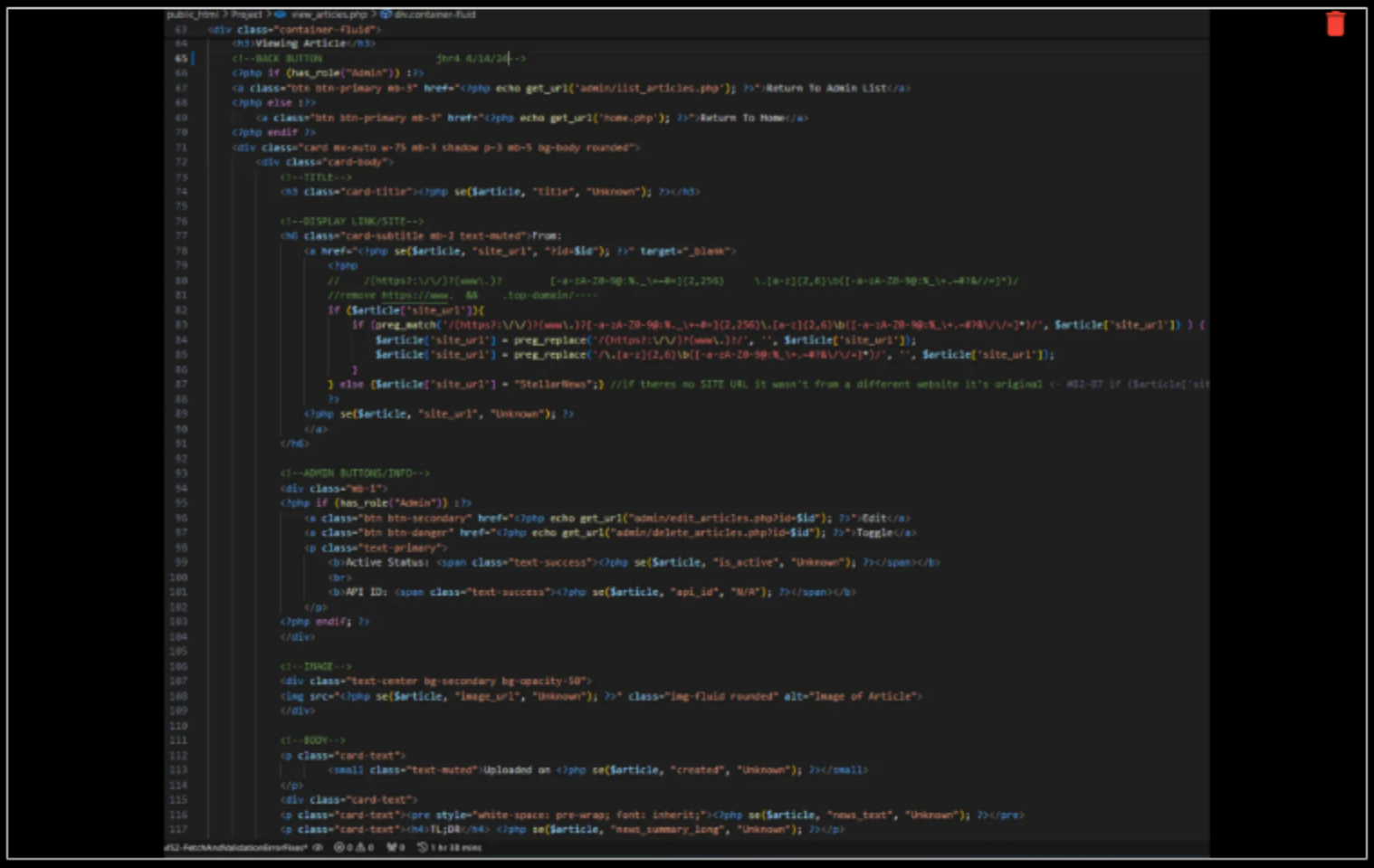
Checklist Items (4)

#1 Show how id is fetched

#2 Show the DB query to get the record

#4 Include ucid/date comments for each code screenshot

#5 Clearly caption screenshots



Shows HTML code on how the data is presented (card Bootstrap) and how links/buttons for edit and delete are shown to only specific users (admins) via a role check.

Checklist Items (1)

#3 Show the code related to presenting the data and showing the links

COLLAPSE

Task #3 - Points: 1

Text: Explain how the page works

Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Provide the high-level steps for handling the DB lookup and presenting the data

Response:

Getting values/data from DB:
For the single view page the article is first fetched from the database by the id. The actual view button is basically taks the primary column id from the data when creating the table and adds it onto the buttons href which goes to the page link followed by ?id="#". This id from the link is treated like GET data and is taken. This id is then used to get data from the database where id=:id basically where the primary column id value matches the provided id and the following columns are requested: api_id, api_timestamp, title, site_url, image_url, news_text, news_summary_long, is_active, created. If there is no value of this primary column in the database table, the user is redirected to the list page again with a message saying invalid id.

Displaying:
As only one article is fetched its easy to display the values. To display it a large card is created with the fields of an article (image, title, body, etc.). To access the data safer echo is used which basically uses the key value pairs of the result (just one article/row of associative array) and echos the values of provided key name in the array. The bootstrap card display is pretty simple and just echos the values in the corresponding divs of title, image, etc. For the image and site_link its echoed in the href/src link. Also for the source/site link (where the article is from), regex is used to strip the link of everything besides the domain and displays that. There's also a return button on the top that goes back to the list page. For the admin the design is a little different and buttons to edit and toggle and additional information is showed (active or not, and api id). This is done by checking if the user has the role admin.

Additionally, when setting the is_active's 0 and 1 value true and false, if the value is false (not active and "deleted") and the user is an non admin, the user gets redirected back to the home page with a message saying it's not available. This is done as the user may have gotten the link from a shared link.

COLLAPSE

Task #4 - Points: 1

Text: Add related links

Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Include the heroku prod link for this page
#2	1	Add the pull request link for the branch related to this feature Note: the link should end with /pull/#. Same pull request shouldn't be used for each feature

URL #1

https://github.com/Jhr-4/jhr4-IT202-008/pull/56

URL #2

https://jhr4-it202-008-prod-af7c79552123.herokuapp.com/Project/view_articles.php?id=293

Edit Data Page (2 pts.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshots of the edit page

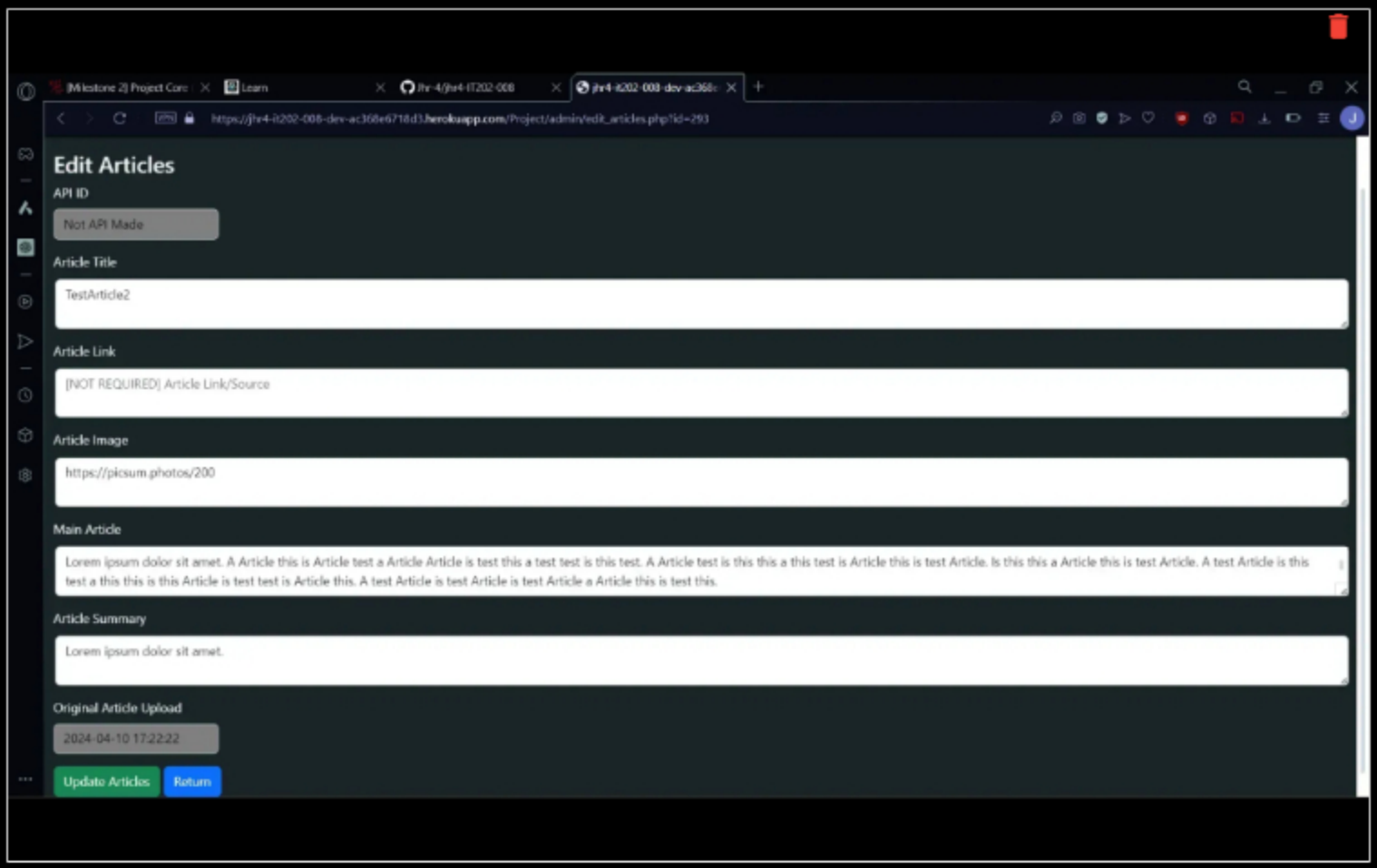
Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Show before and after screenshots of data you'll edit
#2	1	Show examples of validation messages
#3	1	Show an example of successful edit messages
#4	1	Design/Style should be considered (i.e., bootstrap, custom css, etc)
#5	1	Make sure the heroku dev url is visible in the address bar
#6	1	Clearly caption screenshots

Task Screenshots:

Gallery Style: Large View

Small Medium Large



Before screenshot of data I'll edit on website page. Design utilizing simple bootstrap fields. Heroku Dev link on top.

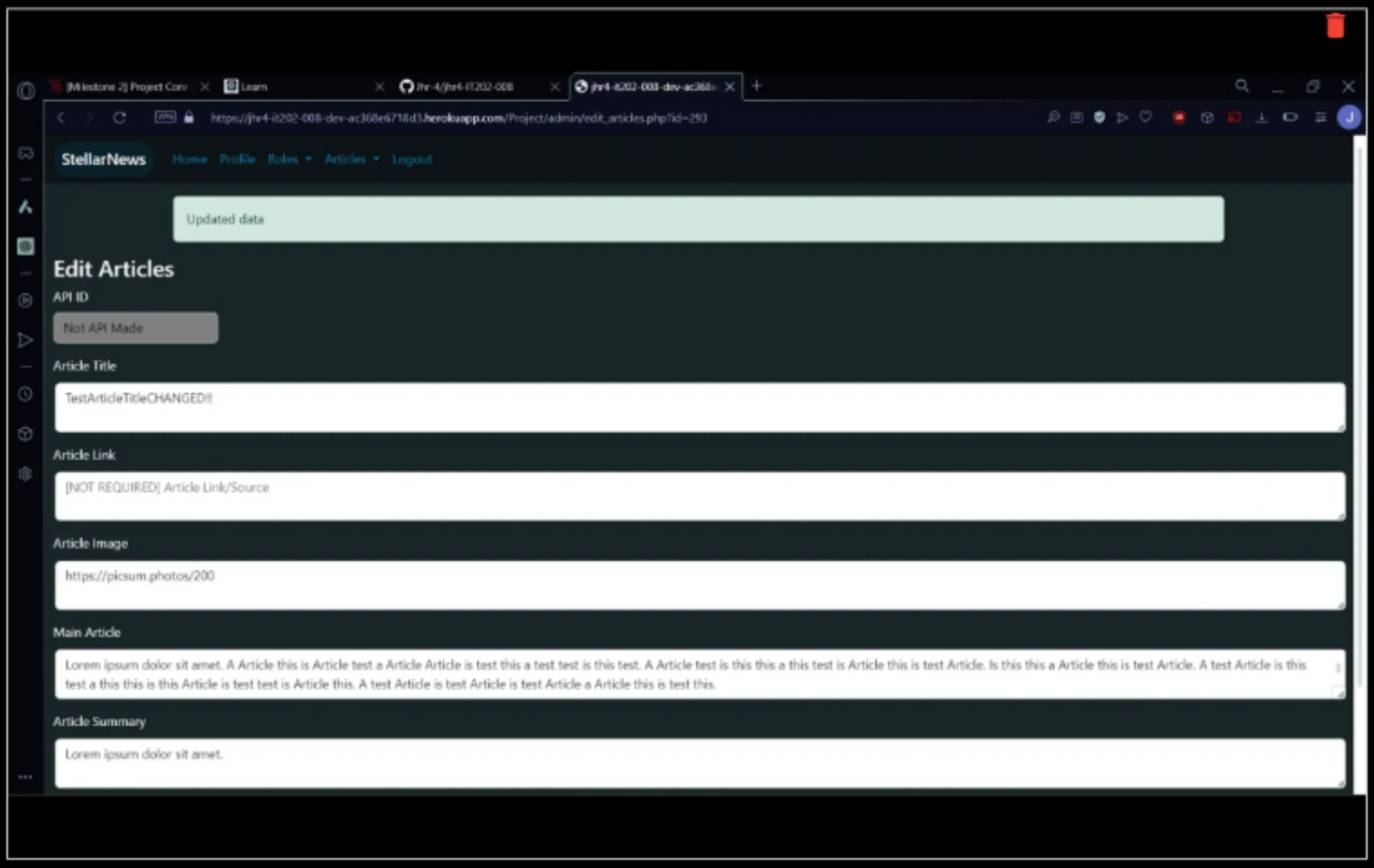
Checklist Items (4)

#1 Show before and after screenshots of data you'll edit

#4 Design/Style should be considered (i.e., bootstrap, custom css, etc)

#5 Make sure the heroku dev url is visible in the address bar

#6 Clearly caption screenshots

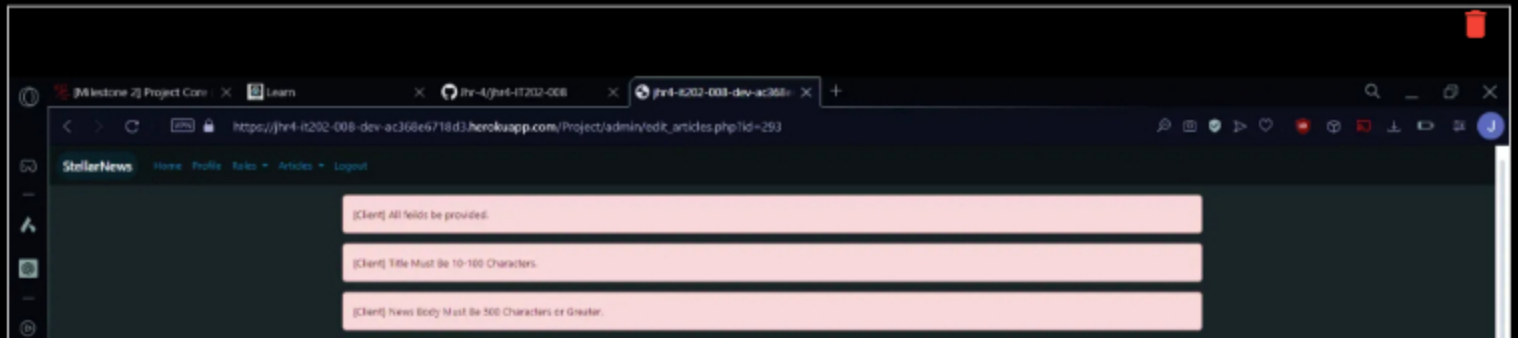


After screenshot of data on website edit page. Successful Message "Updated Data" on top.

Checklist Items (2)

#1 Show before and after screenshots of data you'll edit

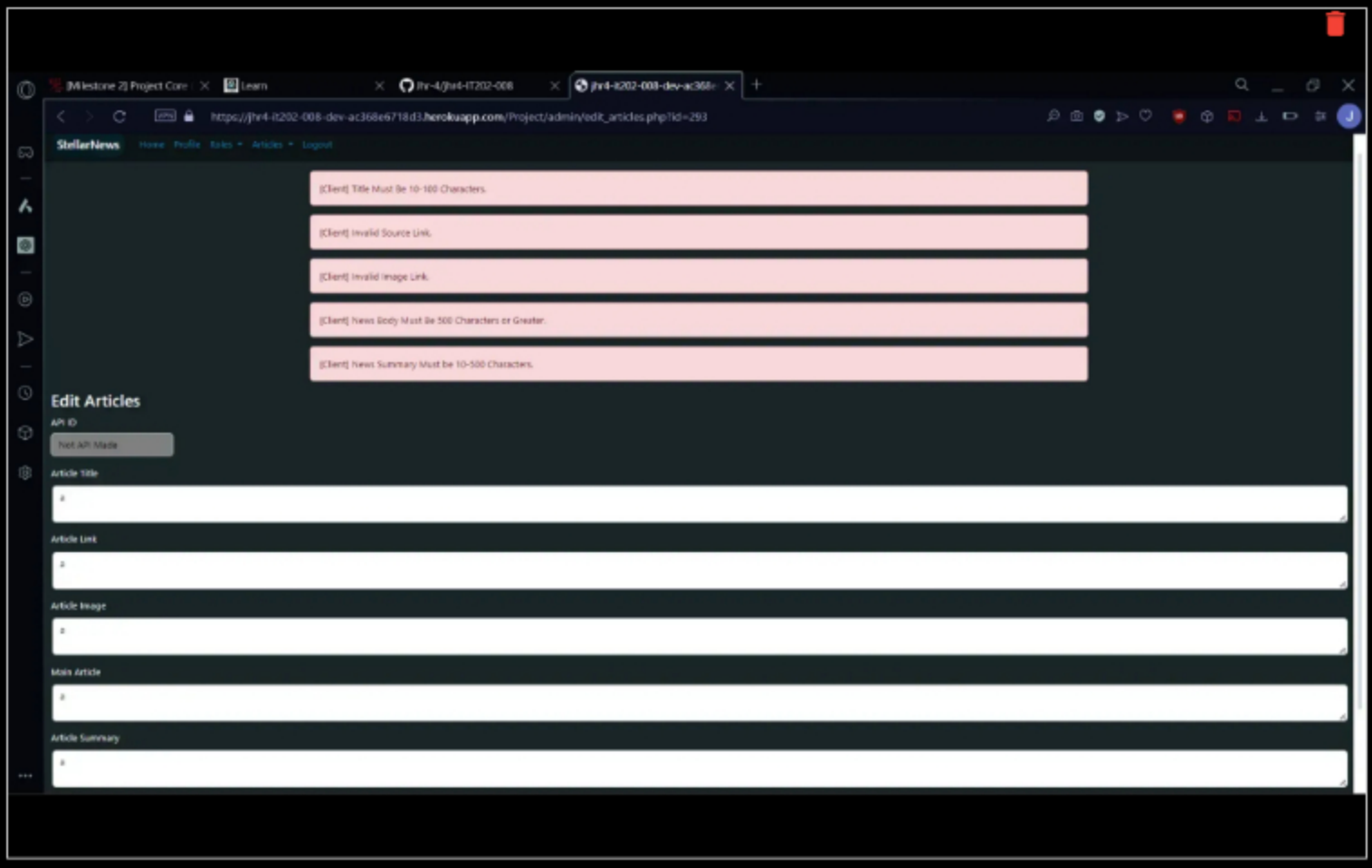
#2 Show examples of validation messages



Client side JS existence validation messages.

Checklist Items (1)

#2 Show examples of validation messages



Client Side JS format validation messages. The same validation is also done in PHP.

Checklist Items (1)

#2 Show examples of validation messages

COLLAPSE

Task #2 - Points: 1

Text: Screenshots of edit page code

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Form should have correct data types for each property being requested
#2	1	Form should have correct validation for each field (HTML, JS, and PHP)
#3	1	Successful edit should have a user-friendly message
#4	1	Any errors should have user-friendly messages
#5	1	Include any other rules like role guards and login checks
#6	1	Include ucid/date comments for each code screenshot
#7	1	Clearly caption screenshots

Task Screenshots:

Gallery Style: Large View

Small Medium Large

public_html > Project > admin > edit_articles.php > ...

```php
<?php //jhr4 4/14/24
//note we need to go up 1 more directory
require(__DIR__ . "/../../../partials/nav.php");

if (!has_role("Admin")) {
    flash("You don't have permission to view this page", "warning");
    die(header("Location: " . get_url("home.php")));
}
?>
```

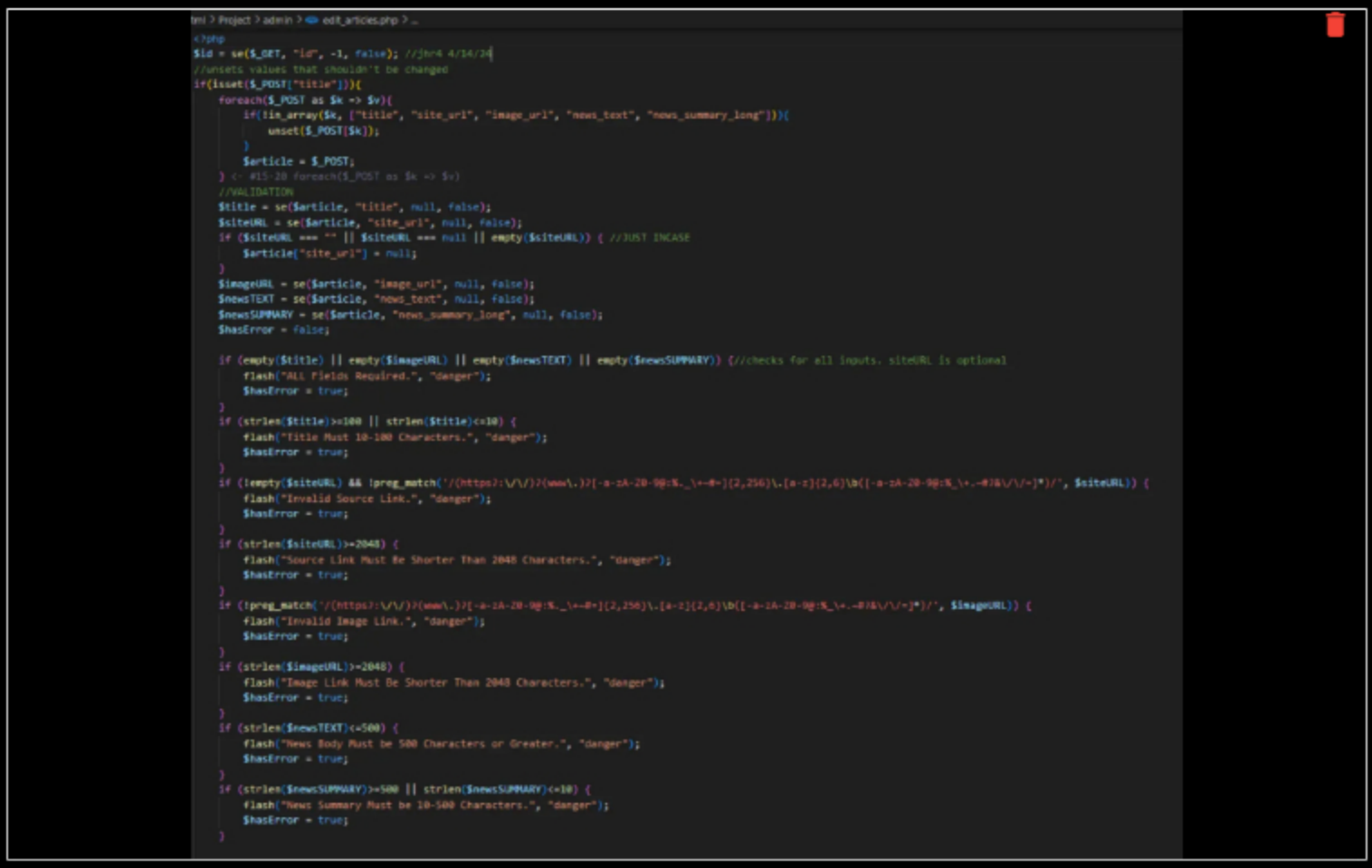
Shows role guard check for admin user this function also checks for if user is logged in. UCID and Date on top.

Checklist Items (3)

#5 Include any other rules like role guards and login checks

#6 Include ucid/date comments for each code screenshot

#7 Clearly caption screenshots

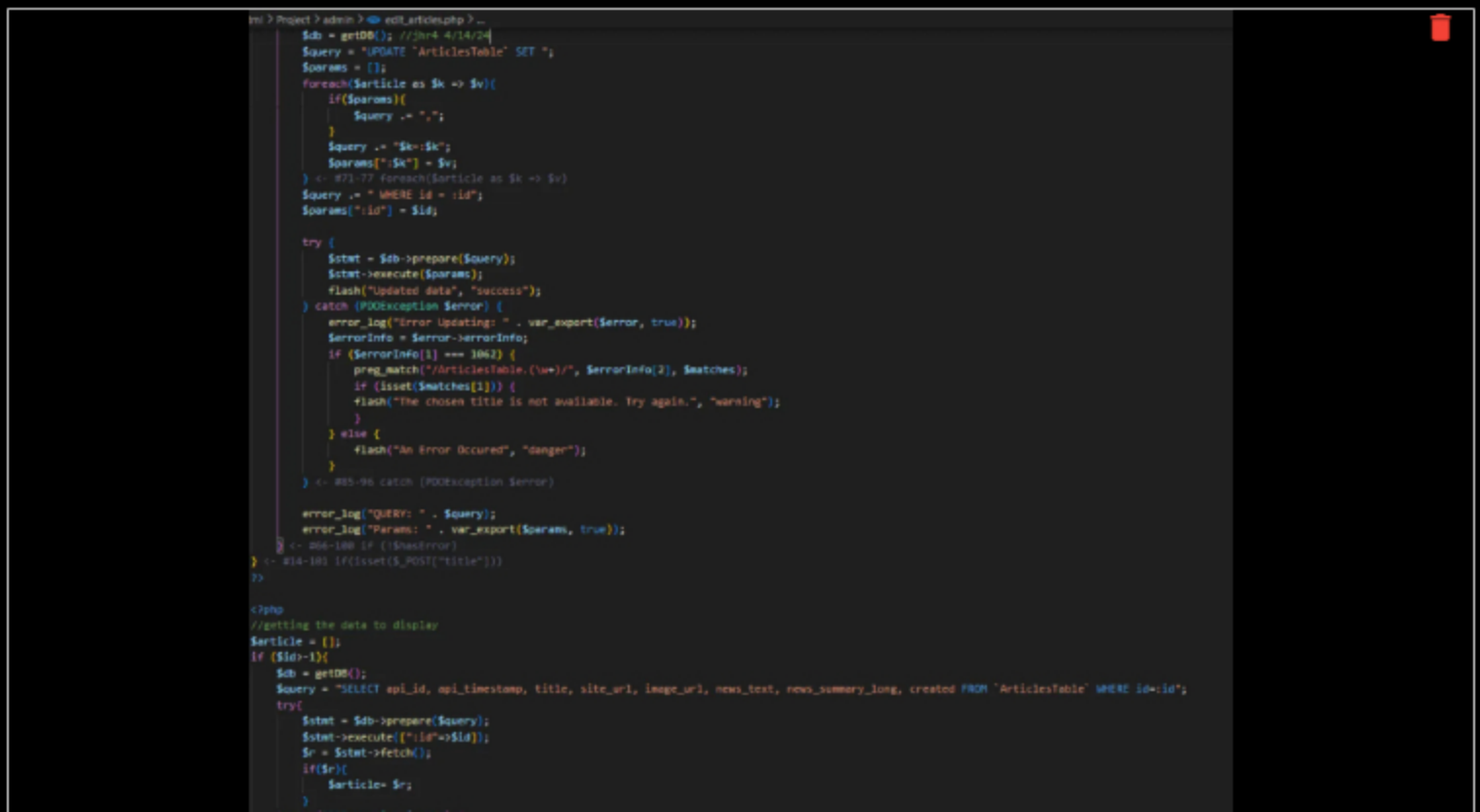


```
tmi > Project > admin > edit_articles.php > ...
<?php
$id = se($_GET, "id", -1, false); //jhr4 4/14/24
//unsets values that shouldn't be changed
if(isset($_POST["title"])){
    foreach($_POST as $k => $v){
        if(!in_array($k, ["title", "site_url", "image_url", "news_text", "news_summary_long"])){
            unset($_POST[$k]);
        }
        $article = $_POST;
    } <- #15-20 foreach($_POST as $k => $v)
    //VALIDATION
    $title = se($article, "title", null, false);
    $siteURL = se($article, "site_url", null, false);
    if ($siteURL === "" || $siteURL === null || empty($siteURL)) { //JUST INCASE
        $article["site_url"] = null;
    }
    $imageURL = se($article, "image_url", null, false);
    $newsTEXT = se($article, "news_text", null, false);
    $newsSUMMARY = se($article, "news_summary_long", null, false);
    $hasError = false;

    if (empty($title) || empty($imageURL) || empty($newsTEXT) || empty($newsSUMMARY)) {//checks for all inputs. siteURL is optional
        flash("ALL Fields Required.", "danger");
        $hasError = true;
    }
    if (strlen($title)>=100 || strlen($title)<=10) {
        flash("Title Must 10-100 Characters.", "danger");
        $hasError = true;
    }
    if (!empty($siteURL) && !preg_match('/(https?:\/\/)?(www\.)?[-a-zA-Z0-9@:%._\+~#=]{2,256}\.[a-z]{2,6}\b([-a-zA-Z0-9@:%_\+.~#?&\/\/=]*)/', $siteURL)) {
        flash("Invalid Source Link.", "danger");
        $hasError = true;
    }
    if (strlen($siteURL)>=2048) {
        flash("Source Link Must Be Shorter Than 2048 Characters.", "danger");
        $hasError = true;
    }
    if (!preg_match('/(https?:\/\/)?(www\.)?[-a-zA-Z0-9@:%._\+~#=]{2,256}\.[a-z]{2,6}\b([-a-zA-Z0-9@:%_\+.~#?&\/\/=]*)/', $imageURL)) {
        flash("Invalid Image Link.", "danger");
        $hasError = true;
    }
    if (strlen($imageURL)>=2048) {
        flash("Image Link Must Be Shorter Than 2048 Characters.", "danger");
        $hasError = true;
    }
    if (strlen($newsTEXT)<=500) {
        flash("News Body Must be 500 Characters or Greater.", "danger");
        $hasError = true;
    }
    if (strlen($newsSUMMARY)>=500 || strlen($newsSUMMARY)<=10) {
        flash("News Summary Must be 10-500 Characters.", "danger");
        $hasError = true;
    }
```

PHP side Validation.

Checklist Items (1)

#2 Form should have correct validation for each field (HTML, JS, and PHP)



```
tmi > Project > admin > edit_articles.php > ...
        $db = getDB(); //jhr4 4/14/24
        $query = "UPDATE `ArticlesTable` SET ";
        $params = [];
        foreach($article as $k => $v){
            if($params){
                $query .= ",";
            }
            $query .= "$k=:$k";
            $params[":$k"] = $v;
        } <- #71-77 foreach($article as $k => $v)
        $query .= " WHERE id = :id";
        $params[":id"] = $id;

        try {
            $stmt = $db->prepare($query);
            $stmt->execute($params);
            flash("Updated data", "success");
        } catch (PDOException $error) {
            error_log("Error Updating: " . var_export($error, true));
            $errorInfo = $error->errorInfo;
            if ($errorInfo[1] === 1062) {
                preg_match("/ArticlesTable.(\w+)/", $errorInfo[2], $matches);
                if (isset($matches[1])) {
                flash("The chosen title is not available. Try again.", "warning");
                }
            } else {
                flash("An Error Occured", "danger");
            }
        } <- #85-96 catch (PDOException $error)

        error_log("QUERY: " . $query);
        error_log("Params: " . var_export($params, true));
    } <- #66-100 if (!$hasError)
} <- #14-101 if(isset($_POST["title"]))
?>

<?php
//getting the data to display
$article = [];
if ($id>-1){
    $db = getDB();
    $query = "SELECT api_id, api_timestamp, title, site_url, image_url, news_text, news_summary_long, created FROM `ArticlesTable` WHERE id=:id";
    try{
        $stmt = $db->prepare($query);
        $stmt->execute([":id"=>$id]);
        $r = $stmt->fetch();
        if($r){
            $article= $r;
        }
```

Top shows code for inserting updates that the user made into the db. Code for displaying user friendly error messages is also shown Line 88-95. Code for displaying user friendly success message is also shown Line 84. Shows code for getting the values of the record (id passed on url/GET) being fetched from the DB to display in the input fields. UCID and Date on top.

Checklist Items (4)

#3 Successful edit should have a user-friendly message

#4 Any errors should have user-friendly messages

#6 Include ucid/date comments for each code screenshot

#7 Clearly caption screenshots

Code for displaying form. HTML Validation existence "required" fields and min/max length. Correct datatypes requested textarea for all of them basically as this allows the user to resize if the article is too long. UCID and Date on Top.

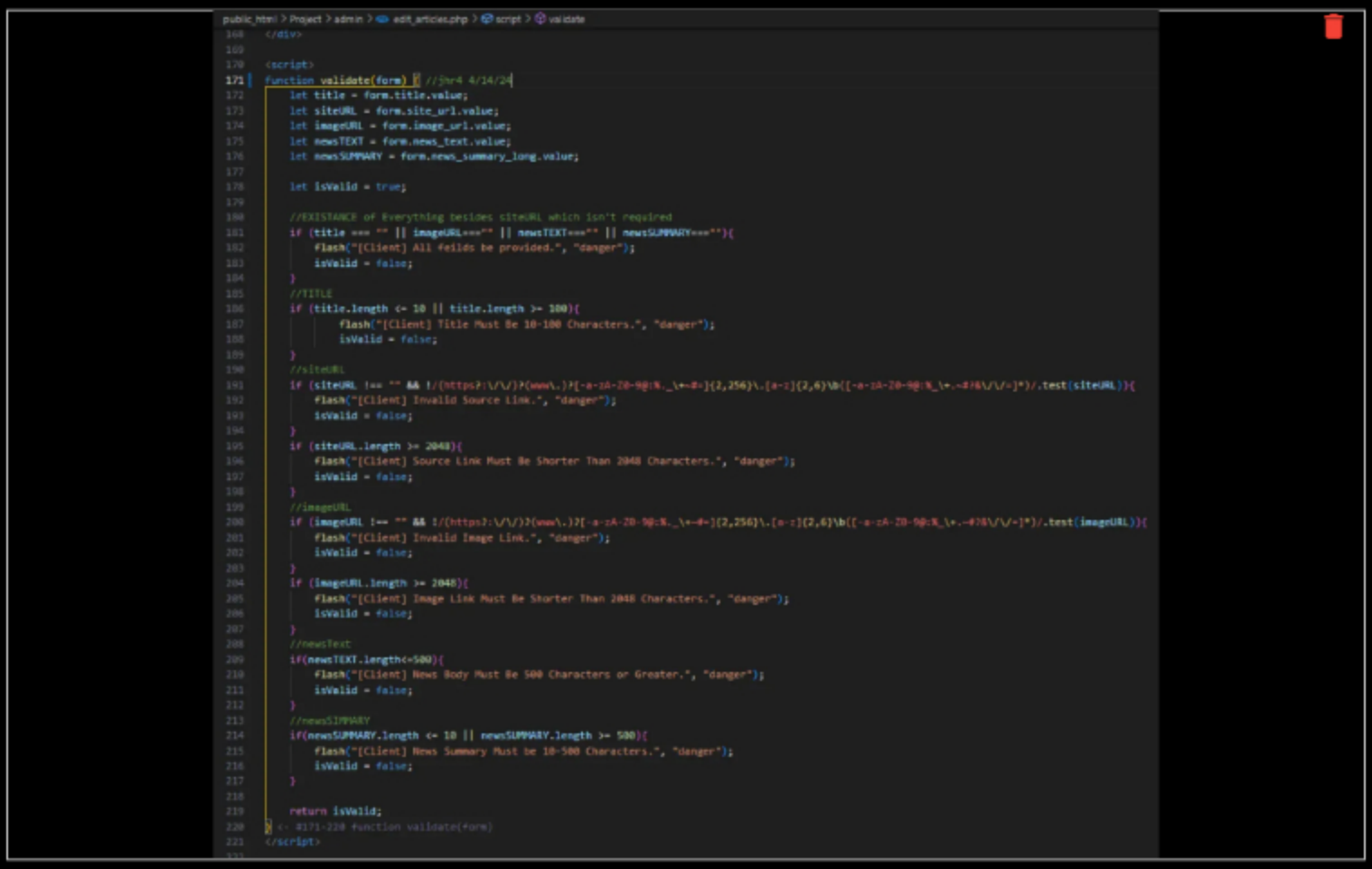
Checklist Items (4)

#1 Form should have correct data types for each property being requested

#2 Form should have correct validation for each field (HTML, JS, and PHP)

#6 Include ucid/date comments for each code screenshot

#7 Clearly caption screenshots



Shows JS Validation. UCID and Date on top.

Checklist Items (2)

#2 Form should have correct validation for each field (HTML, JS, and PHP)

#6 Include ucid/date comments for each code screenshot

COLLAPSE

Task #3 - Points: 1

Text: Screenshot of records from DB

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Show a before and after screenshot of the record
#2	1	Note what differs
#3	1	Clearly caption screenshots

Task Screenshots:

Gallery Style: Large View

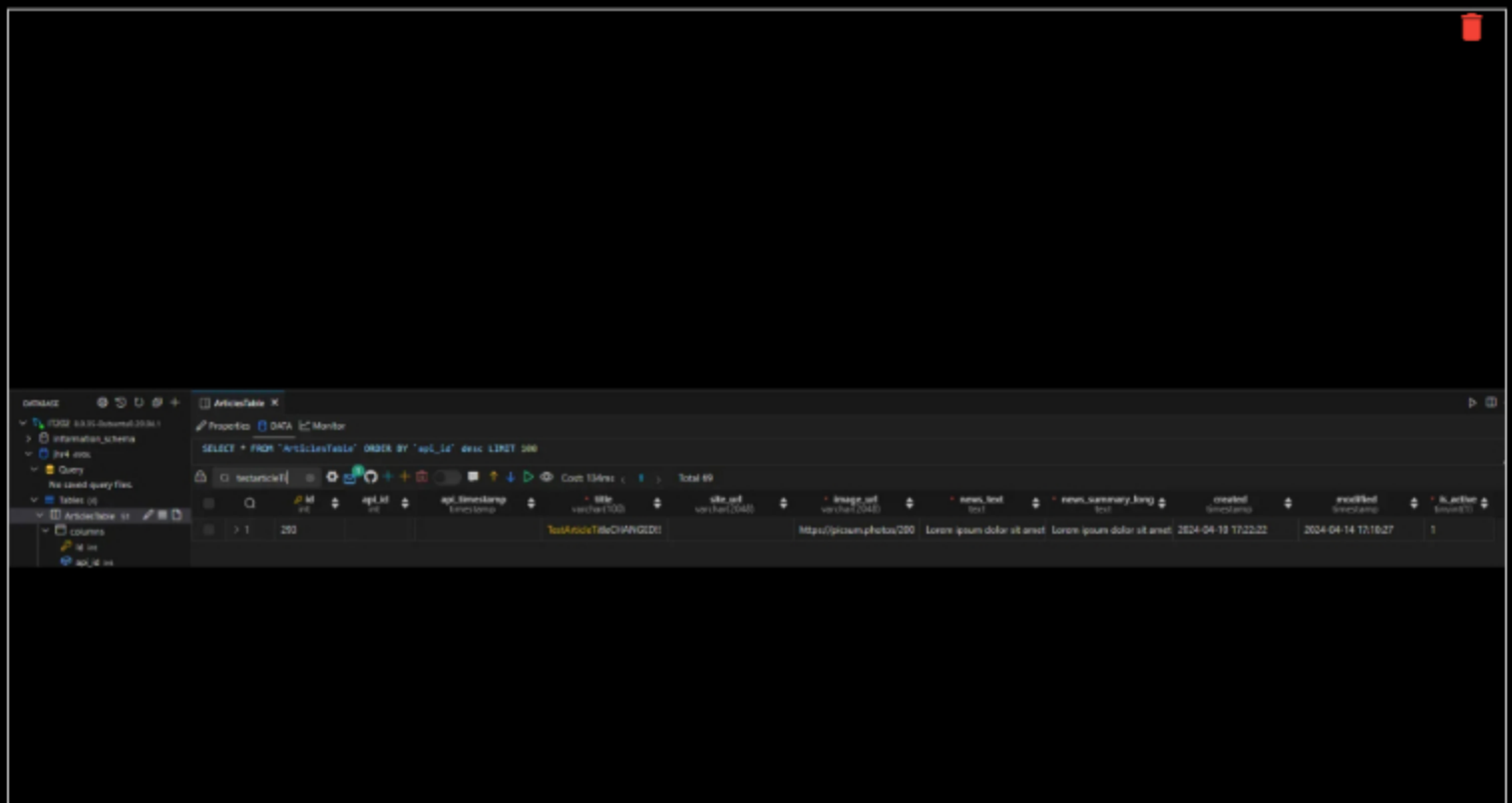
Small Medium Large

Before Image DB: Record 293, The original title is "TestArticle2"

Checklist Items (2)

#1 Show a before and after screenshot of the record

#3 Clearly caption screenshots



After Image DB: Record 293, The changed title is "TestArticleTitleCHANGED!!"

Checklist Items (2)

#1 Show a before and after screenshot of the record

#2 Note what differs

COLLAPSE

Task #4 - Points: 1

Text: Explain the process

Checklist *The checkboxes are for your own tracking

#	Points	Details
#1	1	Provide a high-level step-by-step of how the fetching of the record, populating the form, and the update works and gets changed in your DB
#2	1	Briefly describe the validations for the applicable fields

Response:

Displaying/Fetching:
For the edit page it's similar to the view page for displaying the values. The id is passed through from the button (as explained in the single view section) which then is used to fetch the data from the database table where the primary id key ("id") matches the passed id. If no id, the user is redirected to the list page again. The fetched data corresponds to the selected columns which are: api_id, api_timestamp, title, site_url, image_url, news_text, news_summary_long, and created. To display the fetched values, a form is used with the inputs being named and having the value set to the fetched value. This is done by getting the article data as key value pairs and just checking if the name of the form input matches the key name and then adding the value corresponding the the value="" of the input field. Furthermore, for values that shouldn't be edited (api_id and time created), the fields are "disabled" with the supporting visual css. Furthermore, the fields also have labels that name them for the user to understand.

Submission:
Before clicking submit/update, the values are validated by HTML which checks the length and existence of the required fields. The site_url/source field has no required as the article might be custom made and not form another website/source. However, if the user enters something the length is checked and it must be smaller than 2048 (the limit I set in db table). After the button is clicked there's another client side validation from the JS. The JavaScript also checks for the existence of each field that's required. The format is also checked by checking the lengths of the inputs. Most of the fields should be greater than 10 characters like title and summary but also less than like 100 characters (500 for summary). Furthermore, for the newsText field (body of article) the length has to be greater than 500 to insure its an actual article and not just a random irrelevant line. For things like the image url and site url it's also validated by regex and is checked if it meets the format of a link (contains http(s), www, a domain, top level domain, etc. with the correct symbols/letters/numbers). After this validation, it's again validated by the PHP side in which the same validations occurs again, but in PHP language this is done as the JS or HTML validation can be removed as it's on the user/client end. For each validation if there's a requirement not met the message is flashed on to the screen on what to

change. For JS validation if there's an error it returns false meaning it won't get passed onto the server and php validation. Also this data is passed as a POST response so the GET repone of the id is still available.

After all successful validations, the values are then taken to updating into the database. Here the form irrelevant from keys/values are unset. Basically if the key of the form isn't one of these "title", "site_url", "image_url", "news_text", or "news_summary_long" it will get unset. On the db query this time we are using update as we are trying to edit the article where the id="id". Again, the id is the primary col id and is passed in from the link/GET request. After this the query and params are created via placeholders using the keys from the form after which the params holding the actual form values are executed. If there's an error updating it's shown as an error else, a successful message is shown. The submit also refreshes the page so, upon submit the new data is passed through the fields too which was explained earlier in the displaying part.

An extra thing to note is that this is an admin only page. If the user isn't a admin (doesn't have admin role) they get redirected to the list page.

COLLAPSE

Task #5 - Points: 1

Text: Add related links

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Include the heroku prod link for this page
#2	1	Add the pull request link for the branch related to this feature Note: the link should end with /pull/#. Same pull request shouldn't be used for each feature

URL #1

https://github.com/Jhr-4/jhr4-IT202-008/pull/55

URL #2

https://github.com/Jhr-4/jhr4-IT202-008/pull/57

URL #3

https://jhr4-it202-008-prod-af7c79552123.herokuapp.com/Project/admin/edit_articles.php?id=293

Delete Handling (1 pt.)

COLLAPSE

COLLAPSE

Task #1 - Points: 1

Text: Screenshots related to delete

Checklist

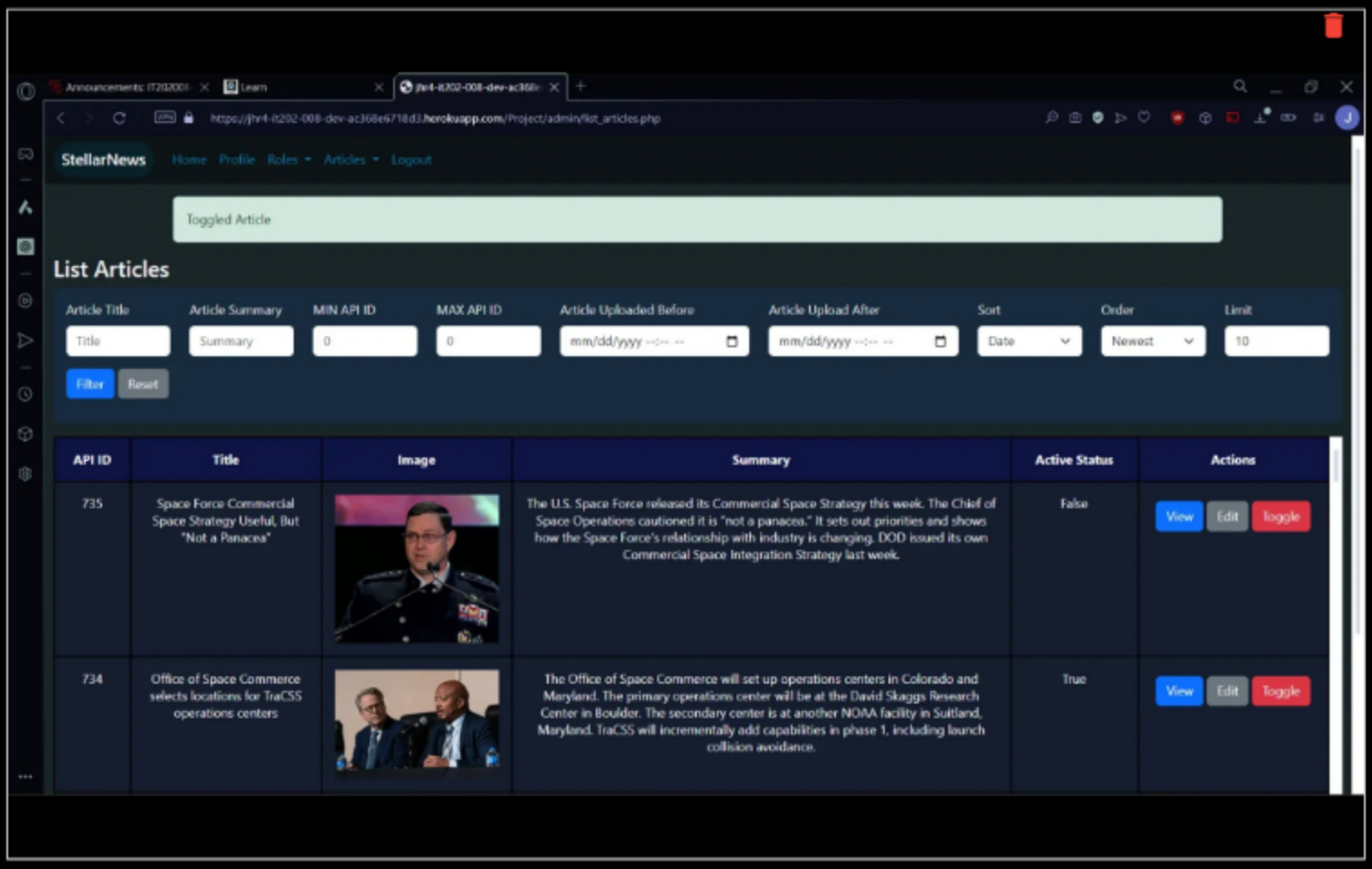
*The checkboxes are for your own tracking

#	Points	Details
#1	1	Show the success message of a delete
#2	1	Show any error messages of a failed delete (like id not being passed)
#3	1	Show the code related to the delete processing

Task Screenshots:

Gallery Style: Large View

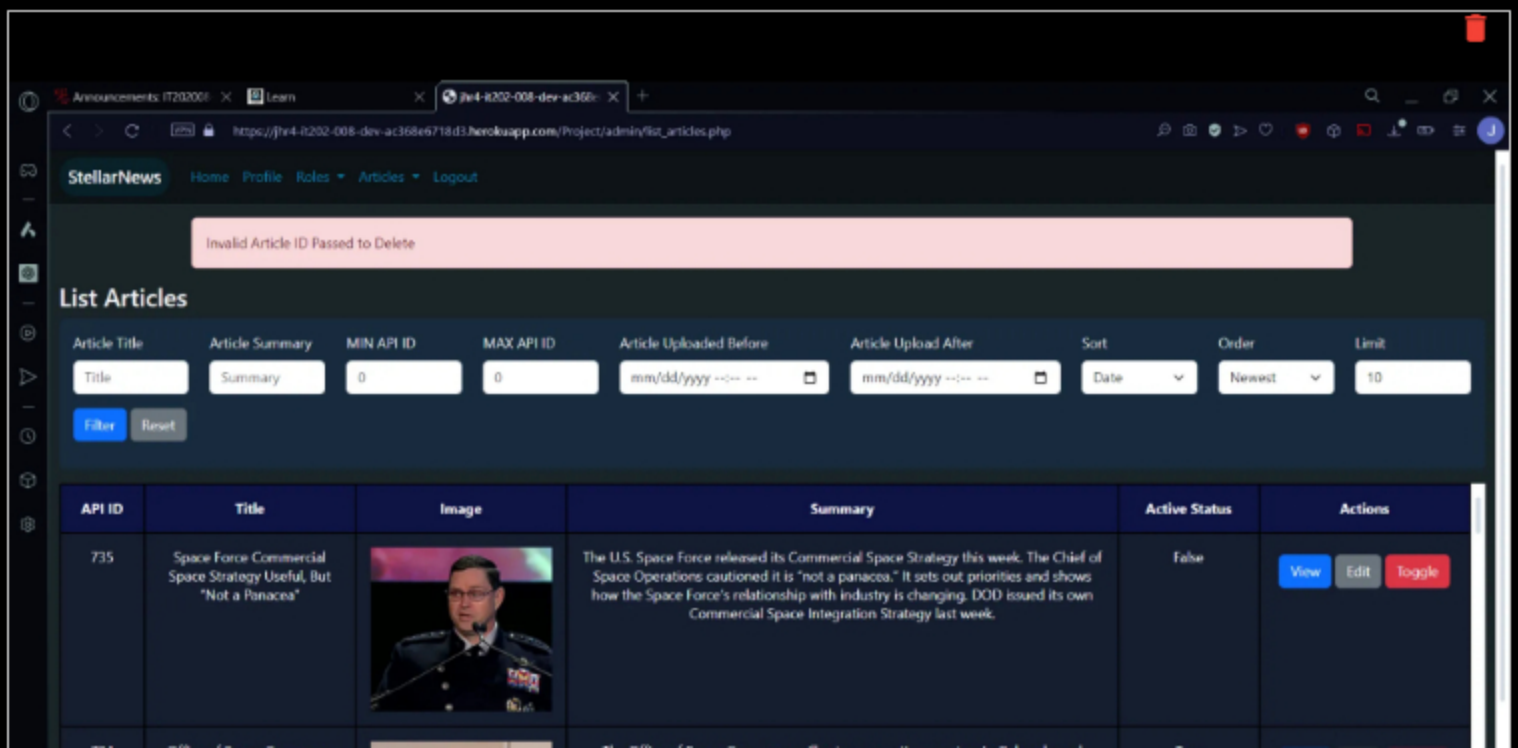
Small Medium Large

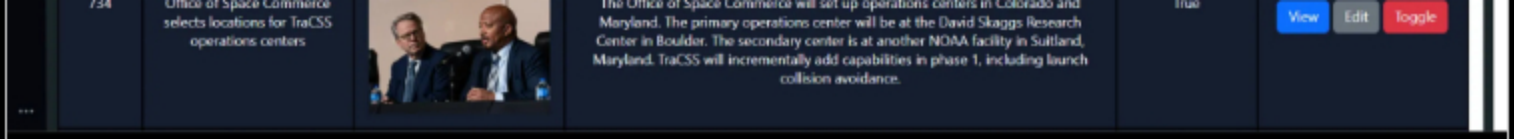


Success message of a delete/toggle.

Checklist Items (1)

#1 Show the success message of a delete





Error message with no ID passed.

Checklist Items (1)

#2 Show any error messages of a failed delete (like id not being passed)

public_html > Project > admin > delete_articles.php > ...

```php
<?php //jhr4 4/14/24
//note we need to go up 1 more directory
require(__DIR__ . "/../../../partials/nav.php");

if (!has_role("Admin")) {
    flash("You don't have permission to view this page", "warning");
    die(header("Location: " . get_url("home.php")));
}

$article_id = se($_GET, "id", -1, false);
if ($article_id < -1){
    flash("Invalid Article ID Passed to Delete", "danger");
    die(header("Location: " . get_url("admin/list_articles.php")));
}
//FOR DELETING/TOGGLING
if (isset($_GET["id"])) {
    if (!empty($article_id)) {
        $db = getDB();
        $stmt = $db->prepare("UPDATE `ArticlesTable` SET is_active = !is_active WHERE id = :aid");
        try {
            $stmt->execute([":aid" => $article_id]);
            flash("Toggled Article", "success");
        } catch (PDOException $e) {
            flash("An error occured", "danger");
            error_log(var_export($e->errorInfo, true));
        }
    } <- #17-27 if (!empty($article_id))
} <- #16-28 if (isset($_GET["id"]))
die(header("Location: " . get_url("admin/list_articles.php")));

?>
```

Code of deleting process. UCID and Date on top.

Checklist Items (1)

#3 Show the code related to the delete processing

COLLAPSE

Task #2 - Points: 1

Text: Screenshots of the data

Checklist

*The checkboxes are for your own tracking

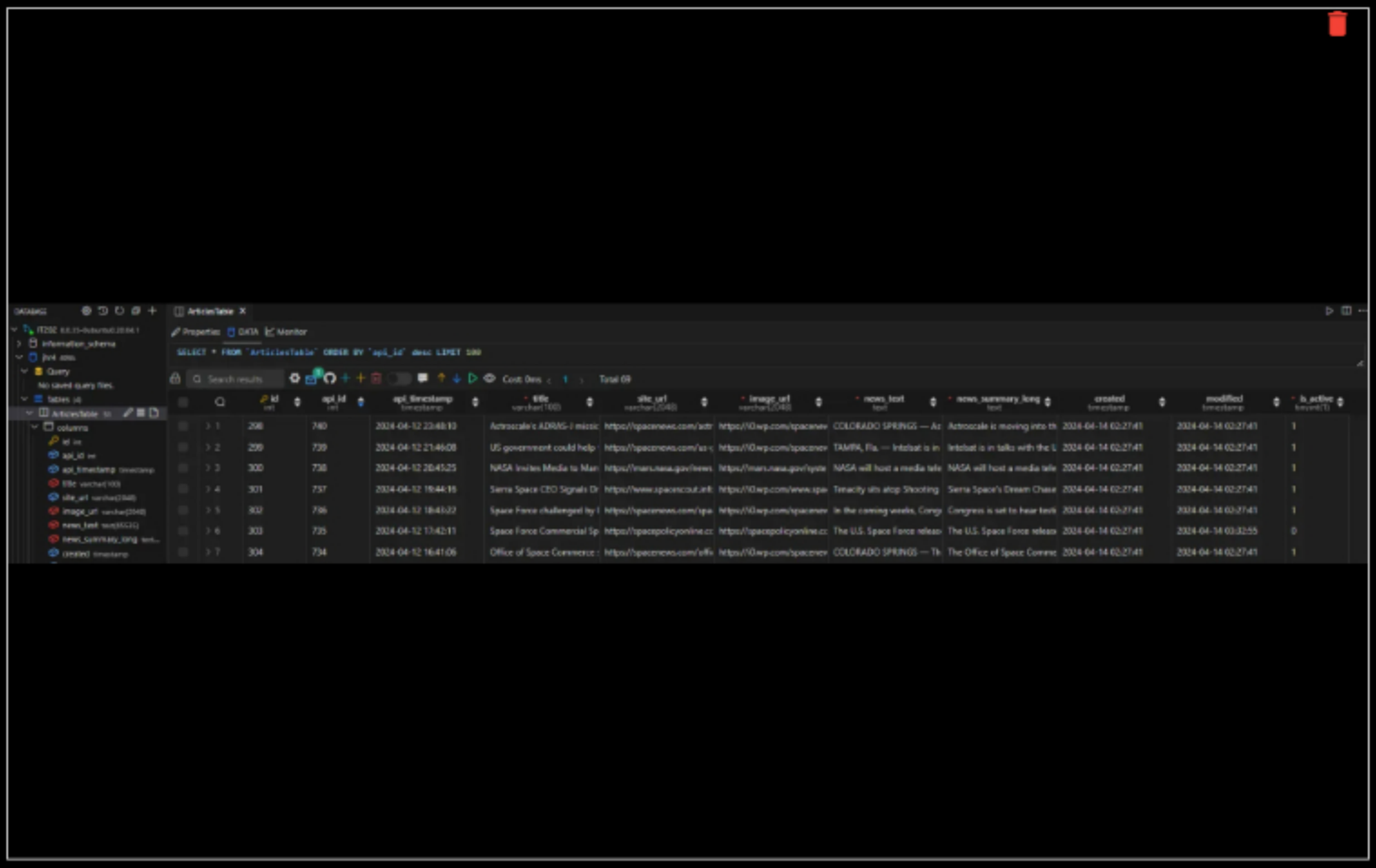
#	Points	Details
#1	1	Show a before and after screenshot of the DB data
#2	1	Note what changed (i.e., record removed or soft delete value changed)

#3	1	Clearly caption screenshots

Task Screenshots:

Gallery Style: Large View

Small Medium Large



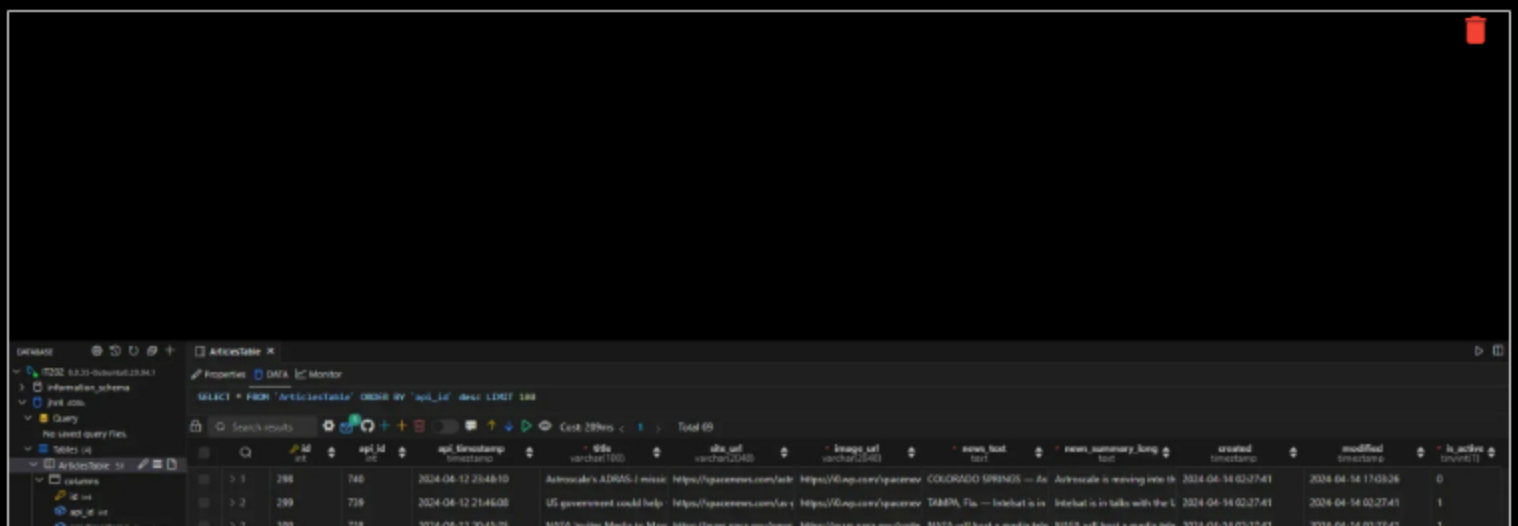
Before Image DB: First record id 298, is_active = 1. This indicates it's active (not deleted).

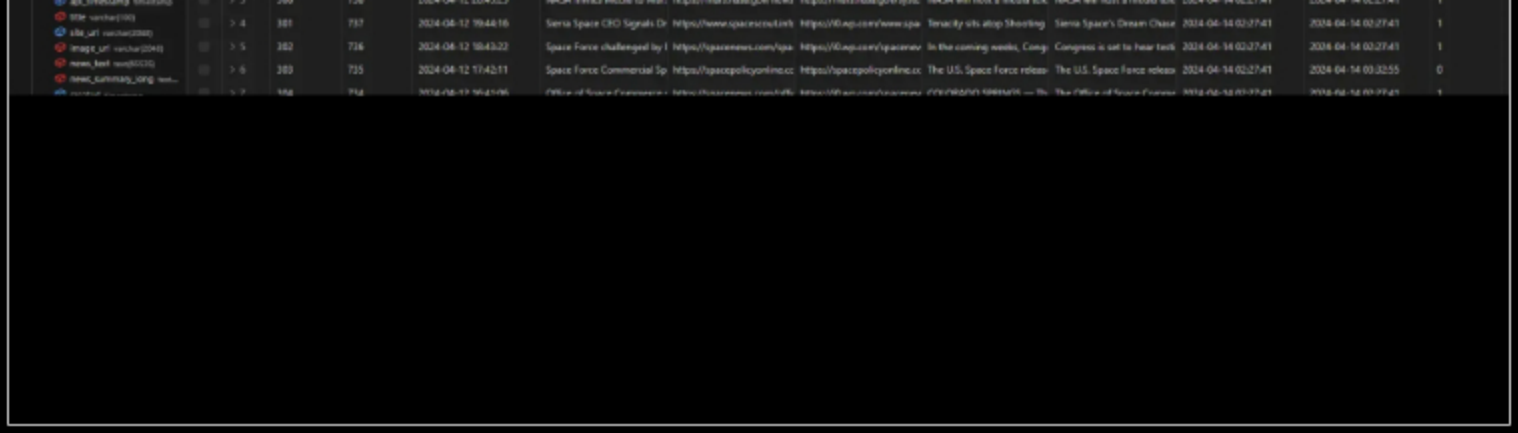
Checklist Items (3)

#1 Show a before and after screenshot of the DB data

#2 Note what changed (i.e., record removed or soft delete value changed)

#3 Clearly caption screenshots





After Image DB: First record ID 298, is_active = 0. Value 0 indicates it's been deleted. Therefore, it's a soft delete just a value change.

Checklist Items (3)

#1 Show a before and after screenshot of the DB data

#2 Note what changed (i.e., record removed or soft delete value changed)

#3 Clearly caption screenshots

COLLAPSE

Task #3 - Points: 1

Text: Explain the delete logic

Checklist

*The checkboxes are for your own tracking

#	Points	Details
#1	1	Is it a soft or hard delete
#2	1	Are there any necessary roles or restrictions? (can only delete their data, can only be done by admin, etc)
#3	1	Provide the high-level steps for handling the DB lookup and handling the delete

Response:

The delete is a soft delete. I added an "is_active" column to the db table which takes values of 0 or 1 (active/deleted or not). The delete is done via a button which redirects the user to the delete page (which also passes in the primary key id which was explained in the view page section). Using this id the delete/update is preformed. When getting the id the database is called and is updated by setting the is_active column to the opposite of is_active. Therefore, if its active (0 returning true) its set to deactivated (1 returning false) where the id is equal to the id that's gotten earlier from the url/get request. After this the page directs the user back to the list page with the message of success or failure based on if there was an error updating or not.

This is an admin page and if a non admin user tries it (without admin role), they get redirected back to the list page.

COLLAPSE

Task #4 - Points: 1

Text: Add the pull request link for the branch related to this feature

Details:

Note: the link should end with /pull/#. Same pull request shouldn't be used for each feature

URL #1

https://github.com/Jhr-4/jhr4-IT202-008/pull/55

URL #2

https://github.com/Jhr-4/jhr4-IT202-008/pull/56

Misc (1 pt.)

COLLAPSE

COLLAPSE

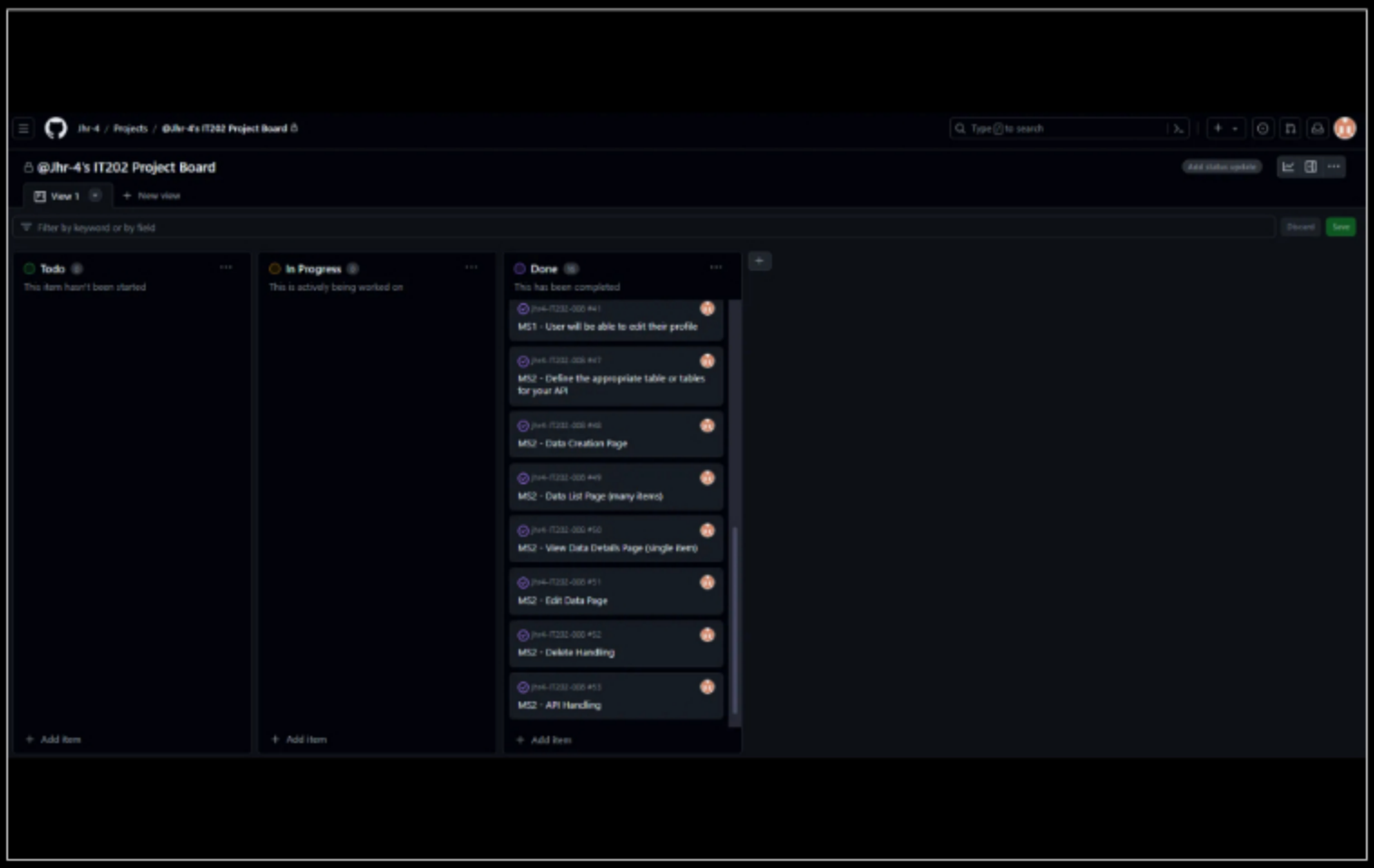
Task #1 - Points: 1

Text: Screenshot of your project board from GitHub (tasks should be in the proper column)

Task Screenshots:

Gallery Style: Large View

Small Medium Large



GitHub Project Board All MS2 items in Done column.

COLLAPSE

Task #2 - Points: 1

Text: Provide a direct link to the project board on GitHub

URL #1
https://github.com/users/Jhr-4/projects/1/views/1

COLLAPSE

Task #3 - Points: 1

Text: Talk about any issues or learnings during this assignment

Response:

I had a few issues with understanding the requirements and some database questions which I got cleared after asking the professor. However, I have one major issue and it was when I was gathering the screenshots today. The API just gave up/died and it was giving a 500, 502, and even 503 error...
From the API:
"messages:"The API is unreachable, please contact the API provider"
info:"Your Client (working) —> Gateway (working) —> API (not working)""
Hopefully this changes before the next milestone..

I learned many things in this assignment especially about working with API's and Database. I learned how to create/fetch data from the API, store it in the DB formatted, read the data from the DB, Update the values in the database (even modify the table), and even how to create soft deletes. Some of it I knew from working with the Roles in Milestone 1, but in this I had to do it myself so I learned more.

COLLAPSE

Task #4 - Points: 1

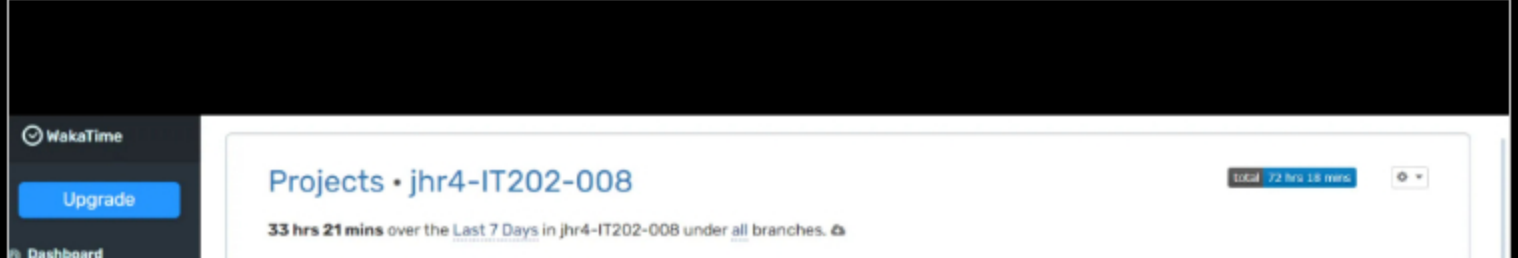
Text: WakaTime Screenshot

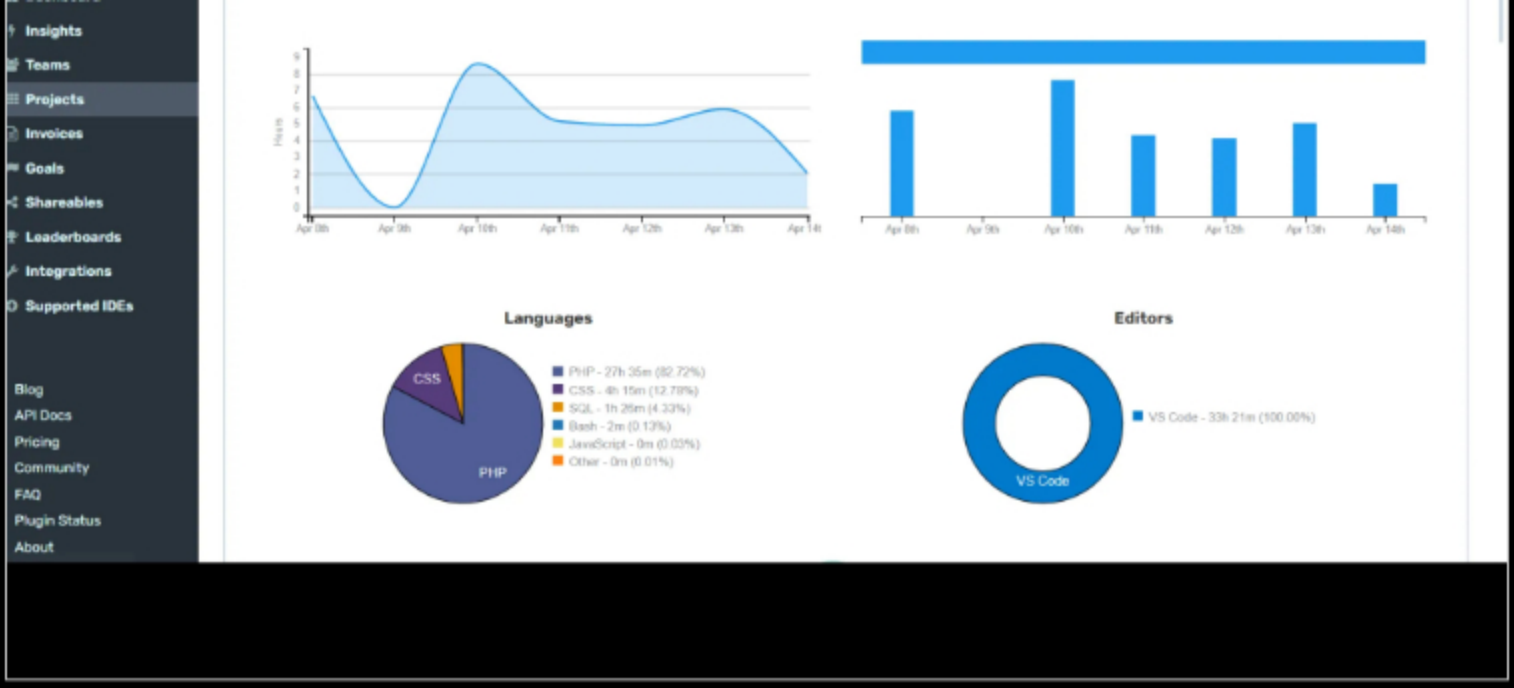
Details:
Grab a snippet showing the approximate time involved that clearly shows your repository. The duration isn't considered for grading, but there should be some time involved

Task Screenshots:

Gallery Style: Large View

Small Medium Large





Waka Time Screenshot (VS Code was just open sometimes...).

End of Assignment